Sistemas Informáticos
Curso 2007 - 2008

Implementación de una plataforma HW para la evaluación de predictores de saltos sobre una arquitectura SPARC v8

Francisco Javier Andrade Irigoyen

Alfredo del Castillo Villalba

Enrique Sedano Algarabel

Dirigido por:

Jesús Javier Resano Ezcaray
Daniel Ángel Chaver Martínez

Dpto.: Arquitectura de Computadores y Automática

Facultad de Informática

Universidad Complutense de Madrid

ÍNDICE

ÍNDICE DE FIGURAS, IMÁGENES Y TABLAS

AUTORIZACIÓN

Los ponentes Francisco Javier Andrade Irigoyen, con DNI 50889978V, Alfredo del Castillo Villalba, con DNI 53388960A, y Enrique Sedano Algarabel, con DNI 51995386E, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos tanto la propia memoria, como el código, la documentación y el prototipo desarrollado.

Francisco Javier Andrade Irigoyen

Alfredo del Castillo Villalba

Enrique Sedano Algarabel

En Madrid, a 4 de Julio de 2008

RESUMEN DEL PROYECTO Y PALABRAS CLAVE

RESUMEN DEL PROYECTO

Los predictores de salto supusieron una gran vía de investigación para la rama de la informática dedicada al estudio e investigación de la arquitectura de los computadores. Su principal objetivo, como el de la gran mayoría de las nuevas ideas en hardware, es la reducción de los tiempos de computación dentro de un procesador, permitiendo realizar la misma carga de trabajo en menos tiempo. Para ello se han ido desarrollando con el tiempo diferentes algoritmos de predicción, cada uno con sus ventajas y sus inconvenientes, pero cuyo resultado en una arquitectura real no puede simularse hasta una vez creada esta, con el consiguiente gasto y riesgo de resultados. Es en esta parte cuando nuestro proyecto resulta destacable. Las herramientas actuales preparadas para simular y medir el éxito o no de un predictor son costosas, además de ser extremadamente lentas (al estar basadas en software y emulación), por lo que conseguir introducir varios de estos predictores en un sistema real y de uso industrial haciendo uso de un sistema hardware reconfigurable como una FPGA resulta ampliamente beneficioso, no ya en coste, sino en tiempo. Para lograr este objetivo hemos implementado una plataforma de evaluación en la que hemos incluido varios predictores que, unidos a un procesador comercial reconocido, nos ha permitido evaluar el rendimiento de cada predictor para diferentes programas. Estos programas, creados expresamente para la medición de datos (comúnmente denominados *benchmarks*) presentan patrones en su diseño bien diferenciados, lo que permite establecer con exactitud y rapidez qué predictor es el más adecuado para un sistema dado teniendo en cuenta tanto el coste HW como las características de las aplicaciones que se van a ejecutar.

ABSTRACT

Branch prediction opened up an important research line for the computer science field dedicated to computer architecture. Its main goal, shared by most novel computer architecture techniques, is to improve the performance of processors, in order to carry out the same workload faster. Many prediction schemes have been proposed and developed in order to achieve this objective,, each of them applying different strategies and with different HW cost. To identify the optimal predictor for a given architecture and set of applications designers must carry out extensive simulations In this context our project presents an important contribution. The current tools used for simulation and test of the efficiency of a predictor are normally based on software emulation. However, this approach provides very slow simulation speed. In this project we propose to use a HW platform based on FPGAs to evaluate very fast several branch predictors in a real industry-used system. To this end, we have implemented an evaluation platform for branch predictors, and we have tested it including representative branch prediction schemes that, connected to a commercial processor, have allowed us to evaluate the efficiency of each predictor for several programs. These programs, developed specifically for performance measurement (commonly known as benchmarks) show clearly distinguished execution patterns, and our platform allow us to accurately identify witch predictor is most suitable for each benchmark taking into account its efficiency and its area cost.

PALABRAS CLAVE

Predictores de salto, evaluación de resultados, FPGA, ISE, ModelSim, VHDL, Virtex II-PRO, SUN SPARC v8, LEON3.

INTRODUCCIÓN

CONTRIBUCION DEL PROYECTO

Cuando nos referimos a la arquitectura de computadores, rama de la ingeniería informática dedicada al estudio y mejora del hardware de un ordenador, hablamos de un objetivo primordial: aumentar la capacidad de procesamiento y velocidad del mismo. Cada nuevo diseño, cada nuevo algoritmo o método está orientado al mismo objetivo concreto: conseguir un máximo aprovechamiento del paralelismo a nivel de instrucción.

La llegada de los procesadores segmentados, superescalares, supersegmentados, así como las actuales tendencias multicore y manycore, supuso un más que notable incremento de rendimiento, gracias precisamente a que explotaban en todo lo posible el citado paralelismo a nivel de instrucción. Sin embargo la presencia de saltos condicionales en el programa en ejecución suponía una importante disminución en el rendimiento. Esta disminución se debía a que dichas instrucciones provocan dependencias de control.

Cuando una instrucción de salto es descodificada, la Unidad de Control bloquea la carga de nuevas instrucciones hasta que se resuelva el salto, lo cual no suele suceder hasta las etapas finales de la ruta de datos (recordemos que en un procesador segmentado, la ruta de datos, o *pipeline*, está dividida en diferentes etapas, una por cada una de las fases de ejecución); como el procesador ha estado varios ciclos sin lanzar ninguna instrucción nueva, hemos perdido nuestra característica de "una instrucción lanzada por ciclo", aparte de tener parte del *pipeline* inutilizado, hasta la resolución del salto.

Es importante tener en cuenta que cuando se habla de la resolución de una instrucción de salto condicional, nos estamos enfrentando a 2 problemas independientes:

- Saber si el salto será tomado o no.
- En caso de que el salto sea tomado, conocer la dirección destino en el menor tiempo posible.

Con el fin de evitar estas fases de espera y evitar las burbujas, surgió el uso de predictores de salto, cuya principal misión es la de intentar anticipar mediante una predicción el resultado de la instrucción de salto (es decir si el salto se tomará o no), así como intentar obtener cuanto antes la dirección efectiva del salto.

El objetivo de todo esto es reducir la penalización producida por los saltos, haciendo prebúsqueda y ejecutando instrucciones sin saber si pertenecen al flujo real del programa. Esto se denomina ejecución especulativa, ya que lanzamos instrucciones a partir de suposiciones sobre el sentido del salto a evaluar.

Visto cómo las instrucciones de salto condicional (que suelen suponer en torno a un 20% de las instrucciones de un programa) provocan graves pérdidas de rendimiento, es obvio que el estudio y refinación de las técnicas existentes de predicción de saltos es un área importante a tratar.

Actualmente existen varios sistemas de evaluación de predictores de saltos, la mayor parte de ellos basados en simuladores software. Uno de los más difundidos a nivel tanto industrial como académico es SimpleScalar [21] [22], herramienta que permite la simulación de varias arquitecturas diferentes (Alpha, PISA, ARM y x86). Sin embargo, este tipo de aproximaciones plantean diversos inconvenientes de velocidad en la obtención de resultados, precisión, y escasez de otros tipos de datos de gran relevancia.

SimpleScalar, simulador que tomaremos como referente, es capaz de ejecutar en torno a 200K Inst/seg en un sistema estándar Pentium 4 a 1.6 GHz en el modo de ejecución Sim-OutOrder. Debido a la lentitud de las simulaciones, al trabajar con aplicaciones grandes con este simulador no se suele ejecutar todo el código de la aplicación en cuestión sino tan sólo

fragmentos discretos de éste, factor que afecta directamente a la precisión de los resultados. A todo esto podemos añadir el hecho de que una simulación de ese tipo no permite conocer factores adicionales como el área de silicio que ocuparía el predictor a evaluar, el retardo que añadiría por ciclo de reloj, o el consumo eléctrico que generaría.

Por ello, debemos descartar la opción de la simulación software como medio de alta fiabilidad para la evaluación de predictores de salto. En su lugar, una aproximación puramente hardware se perfila como un medio de mejores prestaciones para este cometido.

Igualmente, un aspecto muy importante para nosotros es que éste proyecto está orientado, mayoritariamente, al apoyo a la investigación y al desarrollo de nuevas tecnologías, por lo que debería poder ser utilizado sin mayor coste por cualquiera interesado en continuar el trabajo aquí expuesto, o en obtener nuevos resultados a partir del código generado en este proyecto. Por tanto, deseamos establecer como prioridad el desarrollo sobre plataformas de código abierto y licencia de uso público. Con esto en mente, consideramos que la mejor alternativa es trabajar apoyándonos en la arquitectura IEEE-1754 SPARC v8 de Sun Microsystems [16]. De este modo, dado que el código VHDL del procesador utilizado está publicado bajo licencia GPL, y nuestro proyecto es totalmente accesible, el único recargo económico sería el del coste de la plataforma de hardware reconfigurable en la que cargar dicho código. De hecho, esta ambición se ha trasladado a la hora de elegir las herramientas de trabajo para el proyecto, ya que todos ellos se pueden conseguir de manera gratuita, ya sea con licencias freeware como es el caso del Eclipse o el Subversión, o con licencias de estudio, como es el caso del ModelSim o el Xilinx ISE.

Así pues, en este proyecto perseguimos las siguientes metas:

- Obtener una plataforma hardware de alta flexibilidad que permita llevar a cabo la evaluación de múltiples mecanismos de predicción de saltos en un tiempo considerablemente inferior al de los simuladores software utilizados en la actualidad, consiguiendo además una mayor cantidad y calidad de datos útiles para decidir si la política de predicción que se está probando ofrece mejoras con respecto a otras existentes.

- Integrar la plataforma de predicción en el diseño de un procesador real. Para ello, hemos escogido el procesador IEEE-1754 Leon 3 SPARC v8, proporcionado por Gaisler Research [19]. Hemos optado por la utilización de este procesador al tratarse de un sistema de libre distribución, altamente configurable, muy completo en cuanto a funcionalidades y periféricos, y muy generalizado en el ámbito académico, así como en grandes compañías de gran peso en I+D, como la Agencia Espacial Europea (ESA), Actel o Cadence.

- Diseñar una serie de predictores de saltos plenamente funcionales e integrarlos en nuestra plataforma de evaluación para comprobar su correcto funcionamiento.

Es importante, llegados a este punto, hacer hincapié en el hecho de que nuestros objetivos se basan exclusivamente en la predicción del sentido de los saltos a examinar, dejando de lado el cálculo anticipado de la dirección destino de los saltos, área que queda propuesta como posible continuación futura de este proyecto.

OBJETIVOS Y LOGROS REALIZADOS

Desde el principio, buscamos un mecanismo eficiente de cálculo para medir el rendimiento e impacto de diversos predictores de salto en una arquitectura real, de manera que se pueda decidir para qué tipo de aplicación es mejor un predictor que los demás y para cuáles el rendimiento disminuye, así como el tamaño óptimo de cada predictor en función del perfil de ejecución del programa. Con este objetivo en mente, implementamos una plataforma que incorpora todos los contadores pertinentes para el cálculo de la eficiencia de los predictores y le añadimos un módulo de salida por pantalla a través del cual visualizar los valores de éstos registros acumuladores. Así mismo, buscamos siempre realizar los predictores

parametrizables, de modo que pudiésemos reutilizarlos y, sobre todo, cambiar el tamaño de los mismos.

En el proyecto final hemos diseñado e implementado siete predictores, de manera que cada uno es un poco más complejo que el anterior y además hemos tratado de reutilizar componentes dentro de lo posible. Estos predictores son plenamente funcionales, han sido verificados por nosotros mediante simulación, los hemos cargado en la FPGA sobre la que trabajamos, y hemos comprobado que funcionan en tiempo real una vez integrada la plataforma de evaluación con el procesador.

Por otro lado, aprovechamos la mencionada capacidad de modificar la configuración del procesador para eliminar todos los componentes que resultaban innecesarios para nuestros objetivos. Además, encontramos ciertos problemas de sincronización con el módulo de RAM externo, motivo por el cual lo eliminamos también. En total, la superficie de prototipado que nos quedó disponible para incluir nuestros predictores fue de, aproximadamente, el equivalente a un predictor de saltos de 6 Kbytes de tamaño. Podemos considerar los resultados obtenidos para este tamaño máximo como plenamente descriptivos, ya que al tratarse de sistemas empotrados, el tamaño de los predictores rara vez superará esta cota. Si a pesar de ello se quisiese probar el funcionamiento de predictores de mayor tamaño, el único requerimiento sería utilizar una FPGA más grande, que diese cabida a diseños más extensos.

Para comprobar la correcta funcionalidad de los diseños realizados para los predictores, el área que ocupan, así como su rendimiento, sintetizamos y comprobamos todos los módulos desarrollando bancos de pruebas para las herramientas de simulación Post Place & Route en el Xilinx ISE. Estas simulaciones garantizan el mismo resultado que una ejecución en la FPGA final, pero resulta mucho más cómodo trabajar con ellas al poder definirse de forma sencilla todo tipo de bancos de pruebas. Posteriormente, una vez realizada ésta comprobación, comprobamos el correcto funcionamiento de los predictores una vez integrados en la plataforma de evaluación conectada al Leon 3 mediante la ejecución de códigos reales con perfiles de ejecución y saltos conocidos.

Por ultimo, se ha ejecutado un conjunto de benchmarks para sistemas empotrados. Para la carga de éstos programas se propuso ejecutar sobre el procesador un sistema operativo Linux que gestionase la carga y lanzamiento de éstos benchmarks, pero tuvimos que descartar dicha idea ante la falta de memoria RAM suficiente para alojarlo. Tras la ejecución de estas últimas pruebas, las conclusiones a las que hemos llegado se pueden resumir en:

- La plataforma de evaluación implementada funciona en tiempo real sobre la ejecución de una aplicación en el procesador, logrando por ello mejoras de velocidad de hasta tres órdenes de magnitud en la velocidad de obtención de estadísticas con respecto a simuladores software como SimpleScalar. Hemos comprobado además que la introducción de esta plataforma no afecta al tiempo de ciclo propio del procesador, ya que el retardo asociado a ella es inferior al de la ruta crítica.

- Dependiendo del perfil de los programas que se vayan a ejecutar, en ocasiones será irrelevante el predictor a incluir en la ruta de datos, mientras que en otras ocasiones se experimentarán diferencias de hasta el 30% en la tasa de aciertos.

- En general, hemos podido observar que para códigos con patrones sencillos de saltos el rendimiento de los predictores más básicos es aproximadamente tan bueno como el de los más complejos, por lo que la elección de los primeros es más aconsejable en términos de espacio y coste. Sin embargo, para aquellos códigos que presentan una estructura más compleja, el aumento de rendimiento es proporcional a la complejidad del predictor. Del mismo modo, hemos podido establecer el número de entradas en las tablas de los predictores a partir del cual la mejora de rendimiento obtenido no justifica el aumento de las mismas.

ESTRUCTURA DEL DOCUMENTO

El presente documento se encuentra dividido en dos secciones perfectamente delimitadas. En la primera se realiza una aproximación tecnológica de todos los dispositivos, técnicas de reconfiguración y software involucrados en la realización del proyecto. Se describen en detalle las características y funcionalidades de la FPGA utilizada así como las herramientas de diseño empleadas para la realización del proyecto: Xilinx ISE 9.2.04i, ModelSim 6.2g, GRTools y Cygwin Bash Shell. Finalmente, se ofrece una visión general del proyecto, para acabar con el diseño general dentro del procesador que hemos mantenido.

En la segunda parte de la memoria, se describe todo el diseño del proyecto. Se explican todas las alternativas a los predictores de salto implementadas, así mismo comentamos los cambios necesarios en las fuentes de libre distribución del procesador Leon 3, así como su correcta configuración para poder trabajar con él en la FPGA. Asimismo, para comprender mejor el funcionamiento del sistema de saltos, explicamos en un grafico como implementamos el modulo selector de predictores. Tras esto exponemos un análisis comparativo entre varias opciones de hardware, usando varios predictores de salto en comparación con la versión inicial, para poder ver las estadísticas de aciertos de cada uno de ellos. Finalmente, exponemos algunas conclusiones y posibles ampliaciones del proyecto que se pueden realizar como trabajo futuro.

Como apéndice, se muestra un breve manual acerca de la compilación de código C sobre una arquitectura SPARC v8 y un glosario de términos.

APROXIMACION TECNOLOGICA

HARDWARE UTILIZADO

Generalidades sobre las FPGA's

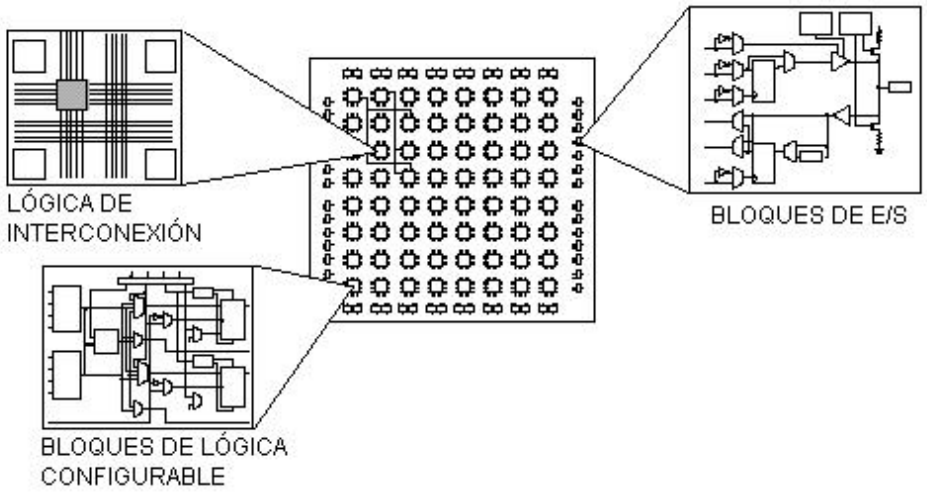
Dentro de las distintas arquitecturas dinámicamente reconfigurables actuales, las FPGAs dominan ampliamente el mercado. La razón principal es que se basan en una tecnología madura con muchos años de desarrollo y sustentada por un amplio conjunto de herramientas de diseño. Por esta razón, el trabajo realizado en este proyecto ha tomado a las FPGAs como punto de referencia a la hora de desarrollar los prototipos y evaluar los resultados.

FPGA es el acrónimo de *Field-Programmable Gate Array*. Es un dispositivo hardware dinámicamente reconfigurable, donde se pueden programar infinidad de diseños digitales distintos.

Las FPGAs son una alternativa a las mask-programmed ASCIs (Application-Specific Integrated Circuit), con las cuales es imposible hacer actualizaciones de diseño sin reemplazamiento de hardware.

Tres son las principales partes de toda FPGA: los Bloques de Lógica Configurables (Configurables Logic Blocks, CLBs), los Bloques de Entrada/Salida (Input/Output Blocks, IOBs) y toda la lógica de interconexión de los elementos funcionales de la FPGA (figura 1). A continuación se tratará brevemente de cada parte.

Figura 1. Bloques funcionales principales de la FPGA



Cada CLB tiene un conjunto de celdas lógicas (Logic Cells, LC). Un LC contiene un generador de funciones implementado, a su vez, con un look-up table que puede programarse para realizar distintas funciones. Además de los LCs, cada CLB contiene lógica que combina los generadores de funciones para permitir funciones de mayor complejidad.

Los IOBs proporcionan el interfaz entre los pines externos y la lógica interna. Cada IOB controla un pin, que puede ser configurado como entrada, como salida o como pin bidireccional. Además cada uno tiene tres registros que comparten la señal de reloj (CLK), pero con distintas señales de capacitación de reloj (CE).

Con la lógica de interconexión concluye la descripción de los elementos funcionales. Su principal característica es que las interconexiones están dirigidas, como el resto de los elementos lógicos, por los valores guardados en las celdas de memoria internas.

Una FPGA debe ser configurada para poder simular un diseño digital, donde configuración se define como el proceso mediante el cual el bitstream (flujo de unos y ceros) de un diseño se carga en la memoria de configuración de la FPGA programando tanto los CLBs como las interconexiones y la E/S. La parte central del proyecto consistió en realizar esta labor, en programar o configurar una FPGA mediante el envío de señales generadas por software.

La FPGA con la que se ha trabajado es la XC2VP30-5 FG676C de familia Virtex-II PRO de Xilinx [1] [17]. La familia Virtex-II se caracteriza por cargar los datos de configuración en celdas internas de memoria estática. Los valores guardados en estas celdas determinan las funciones lógicas y las interconexiones implementadas en la FPGA. El hecho de guardar los valores dentro de estas celdas permite infinitas reprogramaciones del dispositivo. Además en esta FPGA la configuración puede cambiarse en tiempo de ejecución (reconfiguración dinámica) lo que posibilita la realización de un sistema multiprocesador hardware con varias unidades reconfigurables y uno o varios procesadores.

El nombre de la FPGA, XC2VP30-5 FG676C, indica lo siguiente:

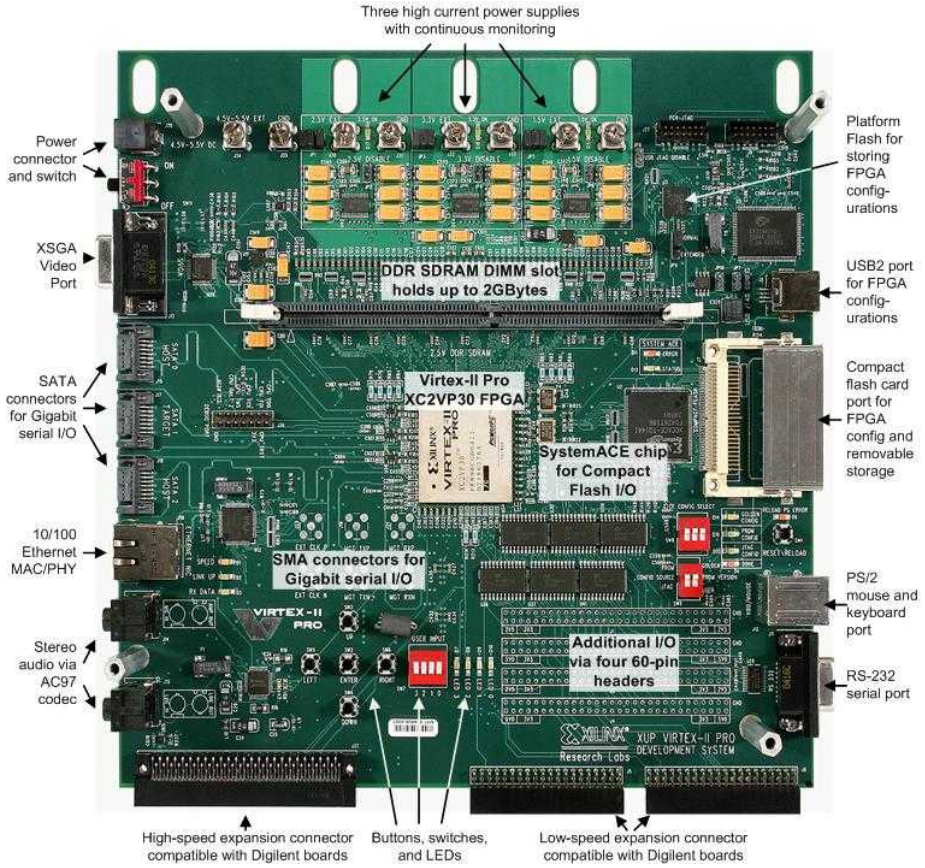
- **Tipo de dispositivo:** XC2VP30, dispositivo 30 de la familia Virtex-II PRO.
- **Grado de velocidad:** 5, estándar.
- **Tipo de embalaje:** FG, Fine Pitch BGA 27 x 27 mm, 1.0 mm ball pitch.
- **Número de pines:** 676.
- **Rango de temperatura:** C, comercial (entre 0º y 85º).

Xilinx Virtex-II Pro

Xilinx es una empresa especializada en el desarrollo de dispositivos programables (FPGAs, CPLDs, etc.) desde 1985. El modelo de FPGA utilizado en este proyecto es el Virtex-II Pro, desarrollado por Xilinx en 2004. A continuación se enumeran algunas de sus características principales:

- Arquitectura lógica programable
  - Tecnología de 130nm y 9 capas de cobre
  - Desde 3000 hasta 99000 celdas lógicas
  - Velocidad de reloj superior a los 400MHz
  - Alto rendimiento y bajo consumo
- Escalabilidad. Hasta 11 formatos.
- Características avanzadas
  - Memoria distribuida y empotrada (Flash)
  - Gestión de relojes digitales
  - Reconfiguración de la FPGA total/parcial en función de la actualización de los productos del mercado
  - Tecnología XCITE que permite mejorar la integridad de la señal y reducir espacio
- Conectividad
  - Hasta 20 *serial transceivers* (RocketIO) *full-duplex* (de 622Mbps hasta 3.125Gbps)
  - Conexiones a 100MHz mediante LVDS
- Procesamiento avanzado
  - Dos procesadores empotrados IBM PowerPC 405 a 400MHz
- Herramientas de desarrollo
  - Herramientas para la programación de la FPGA

Figura 2. Xilinx XUP Virtex II – Pro



VHDL

El VHDL (*Very High Speed Integrate Circuit Hardware Description Language*), es un lenguaje de descripción y modelado, diseñado para describir la funcionalidad y la organización de sistemas *hardware* digitales, placas de circuitos, y componentes.

La finalidad del modelado es la simulación. La sintaxis amplia y flexible del lenguaje VHDL permite tanto el modelado estructural como el modelado funcional de circuitos. En el primer caso, se describe el circuito indicando los componentes y las conexiones que lo componen (lo cual requiere un conocimiento detallado del circuito). En el segundo caso, se describe el circuito indicando lo que hace y cómo funciona, es decir, describiendo su comportamiento (sin necesidad de conocer su estructura interna).

Esta segunda metodología de modelado resulta muy interesente desde el punto de vista del diseño de sistemas digitales. Más aún teniendo en cuenta que hoy en día otra de las aplicaciones del lenguaje VHDL, con una gran demanda de uso, es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se desciende verticalmente por los niveles de la jerarquía de diseño hasta llegar a una implementación más detallada, menos abstracta. Puesto que el VHDL fue inicialmente concebido para el modelado de sistemas digitales, su utilización en síntesis no es inmediata. Sin embargo, la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

Así pues, VHDL permite diseñar, modelar y comprobar un sistema desde un alto nivel de abstracción hasta el nivel de definición estructural de puertas lógicas. Además, siguiendo ciertas guías para síntesis, permite la implementación de diseños a nivel de puertas lógicas. Al estar basado en un estándar (IEEE Std. 1076-1987) reduce errores de comunicación y problemas de compatibilidad. Finalmente, dada su característica de modularidad, permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

Los componentes de un proyecto VHDL son los siguientes:

- **Entity**: Es el más básico de los bloques de construcción en un diseño. Una entidad VHDL especifica el nombre de la entidad, sus puertos, e información relacionada con ella. Todos los diseños son creados usando una o varias entidades. La entidad describe la interfaz en el modelo VHDL.

- **Architecture**: La arquitectura describe la funcionalidad esencial de la entidad y contiene los estados que modelan el funcionamiento de ésta. Una entidad puede tener varias arquitecturas.

- **Configuration**: Permite unir la instancia de un componente a la pareja entidad-arquitectura. Describe el comportamiento a utilizar para cada entidad.

- **Package**: Es una colección de los tipos de datos y subprogramas usados comúnmente en un diseño. Las librerías forman parte de los *packages*.

Procesador GPL IEEE-1754 Leon 3 SPARC v8

Introducción

El procesador de libre distribución IEEE-1754 Leon 3 SPARC v8, implementado por la empresa Gaisler Research, es un modelo sintetizable en VHDL de un procesador de 32 bits, basado en la arquitectura SPARC v8. Este modelo es altamente configurable y particularmente adecuado para diseños de sistemas empotrados (System-On-a-Chip, SOC). Todo el código está disponible mediante la licencia GNU GPL, permitiendo uso libre e ilimitado para investigación y educación. El Leon 3 también se encuentra disponible en una versión de bajo coste comercial, permitiendo ser usada en cualquier aplicación comercial por una fracción del coste de los núcleos IP comparables.

Presenta las siguientes características:

- Instrucciones basadas en la arquitectura SPARC v8 con extensiones v8e.
- Pipeline avanzado de 7 etapas.
- Multiplicador, divisor y unidades MAC hardware.
- Posibilidad de uso de la librería GRFPU de Gaisler, de alto rendimiento, totalmente segmentada conforme al IEEE-754.
- Caches de datos e instrucciones separadas, basadas en la arquitectura Harvard, con snooping.
- Caches configurables: de 1 a 256 kbytes/cache, de 1 a 4 caches y con reemplazamiento aleatorio, LRR o LRU.
- MMU de referencia SPARC (SPMMU) con TLB configurable.
- Interfaz de bus AHB AMBA 2.0.
- Avanzado sistema de soporte para debug on-chip, con soporte para instrucciones y buffer de traza de datos.
- Soporte para multi-procesadores simétricos (SMP).
- Modo de suspensión y clock gating.
- Diseño robusto y totalmente síncrono basado en un solo evento de reloj.
- Hasta 125 MHZ en una FPGA y 400 MHZ en las tecnologías ASIC de 0,13 um.
- Dispone de tolerancia a fallos y una versión de prueba de SEU para aplicaciones espaciales.
- Gran cantidad de herramientas software, compiladores, kernels, simuladores y monitores para debug.

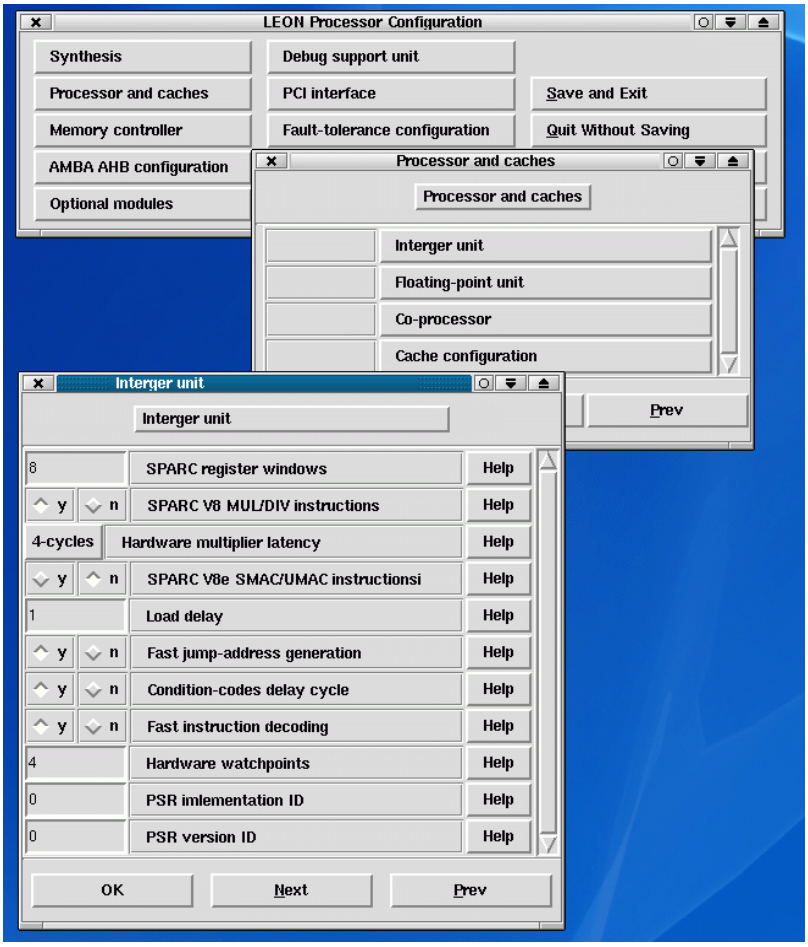
El procesador Leon 3 se distribuye como parte de la librería GRLIB IP [14] [15], en hemos usado la version grlib-gpl-1.0.17-b2710, que permite la integración en diseños SOC complejos. Esta librería también incluye un diseño multi-procesador configurable del Leon 3, con un máximo de 4 CPU's y un gran rango de bloques periféricos on-chip.

Configuración

El procesador Leon 3 es totalmente configurable gracias al uso de archivos VHDL genéricos, y no depende de un único paquete de configuración. Es incluso posible instanciar varios procesadores en el mismo diseño con diferentes configuraciones. Las platillas de diseños del LEON· pueden configurarse usando una herramienta gráfica construida sobre Linux, pero utilizable mediante emuladores como CygWin o MSys en Windows. Esto permite a nuevos usuarios definir rápidamente una configuración personalizada y adecuada en cada caso.

La herramienta de configuración no solo cambia el procesador en si, además cambia otros periféricos on-chip tales como los controladores de memoria o las interfaces de red.

Figura 3. Ejemplo del configurador del Leon 3



Síntesis

El procesador Leon 3 puede ser sintetizado usando las herramientas de síntesis comunes, tales como Synplify, Synopsys DC y Cadence R. El nucleo puede funcionar a las velocidades anteriormente mencionadas, distinguiendo entre FPGA y tecnologías ASIC. El área del núcleo (pipeline, controladores de cache y unidades de multiplicación y división) requiere únicamente entre 20 y 25 mil puertas o 3500 LUT, dependiendo de la configuración. El procesador puede ser sintetizado con Xilinx XST y Altera Quartus, tanto por consola usando scripts o mediante las interfaces gráficas que estos programas presentan.

Desarrollo software

Al estar basado en SPARC v8, los compiladores y kernels para esta arquitectura pueden usarse para el Leon 3 (los kernels necesitaran el LEON bsp). Para simplificar el desarrollo de software, Gaisler Research distribuye BCC, un sistema compilador de código C, C++ basado en gcc y la librería empotrada Newlib. BCC tambien incluye soporte para interrupciones y una librería de Pthreads. Para aplicaciones multi-thread y/o multi-procesado existe una implementación del sistema eCos para el LEON, un kernel de tiempo-real. Asimismo y para aplicaciones industriales existen ports de Nucleus, VxWorks (5.4 y 6.3) y ThreadX.

Existe un soporte Linux para el Leon 3, gracias a una versión especial del sistema SnapGear Linux para sistemas empotrados. SnapGear consiste en un paquete completo, que contiene kernel, librerías y códigos de aplicaciones para un rápido desarrollo de sistemas empotrados Linux. Esta implementación para el Leon 3 soporta las configuraciones tanto MMU

como no MMU del procesador, así como las unidades opcionales de multiplicación y división v8 y la unidad de punto flotante (FPU). Estas unidades pueden descargarse de la página oficial de Gaisler Research, pero para el caso de FPU, existen varias alternativas disponibles. La última versión de SnapGear permite incluso el uso de multi-procesadores.

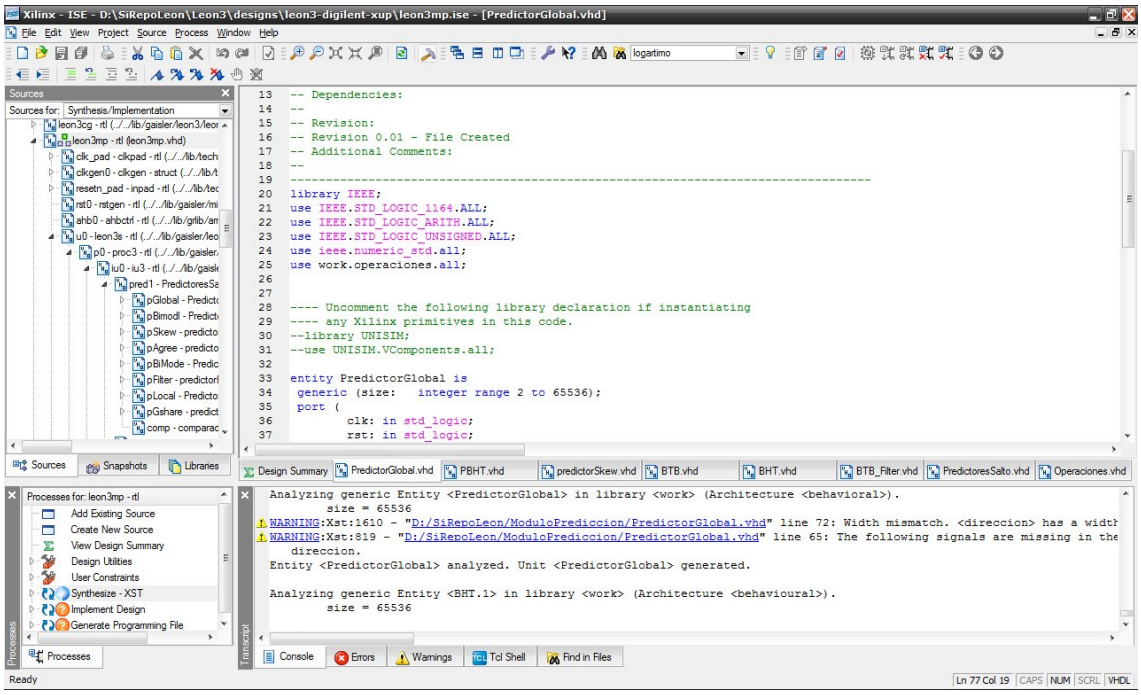
SOFTWARE UTILIZADO

Para el desarrollo de la implementación en VHDL tanto de las modificaciones necesarias al Leon 3 y los diferentes predictores de saltos, así como su posterior testeo, se utilizaron los programas Xilinx ISE 9.2i y ModelSim 6.2g respectivamente. Para configurar el Leon 3 y adecuarlo a la tecnología de la FPGA VirtexII-Pro fue necesario el uso del emulador de entorno de Linux para Windows, CygWin. Por ultimo, para lograr ejecutar programas de benchmark con los que medir el funcionamiento de los predictores, así como compilar estos programas para la arquitectura SPARC v8 necesitamos usar el paquete de herramientas GRTools, de la empresa Gaisler Research.

Xilinx ISE 9.2i

El entorno de desarrollo ISE de Xilinx posee un aspecto similar al de los entornos de programación actuales como puede ser Visual Basic o Visual C, es decir, posee diversas ventanas para la visualización de tareas específicas sobre cada una de ellas. En este caso existen cuatro tipos de ventanas (figura 3):

Figura 4. Pantalla principal del entorno Xilinx ISE



1. Ventana de ficheros fuente. En esta ventana se muestran los ficheros fuentes utilizados en el diseño y las dependencias entre ellos. También es aquí donde se elige el tipo de dispositivo donde se desea almacenar el diseño. Esta ventana posee diversas solapas para visualizar diferentes tipos de información relativa a las fuentes de diseño empleadas.
2. Ventana de Procesos. Esta ventana muestra todos los procesos necesarios para la ejecución de cada etapa de diseño. La lista de procesos se modifica dinámicamente dependiendo del tipo de fuente seleccionado en la ventana de ficheros fuente.

3. Ventanas de edición. Al hacer doble clic sobre un fichero fuente de la ventana de ficheros fuente se abre una ventana de edición para modificar el fichero (en caso de lenguaje VHDL), o bien se ejecuta el programa que permite editar el diseño (en caso de diseños esquemáticos o máquinas de estado).
4. Ventana de información, situada en la parte inferior. Muestra mensajes de error, aviso o información emitidos por la ejecución de los programas de compilación, implementación, etc.

Tanto en la ventana de procesos como en la de ficheros fuente es posible modificar las opciones de cada elemento a través del botón derecho del ratón, o bien a través de los menús del entorno de diseño; estos menús se modifican dependiendo del tipo de selección realizada en las ventanas de ficheros fuente y de procesos. Cada elemento mostrado en la ventana posee un icono diferente dependiendo del tipo de acción o fichero de que se trate, por ejemplo, nos indica si un elemento es un documento de texto, una acción para ejecutar en el entorno ISE o una acción para ejecutar por un programa adicional como puede ser ModelSim a la hora de simular. También muestra información sobre el estado que ha dado resultado tras la ejecución del proceso, es decir, si ha sido satisfactorio, si ha tenido errores, o ha tenido avisos. Las imágenes de la siguiente figura muestran un ejemplo de los diferentes tipos de información mostrados en estas dos ventanas. Para la ventana de ficheros fuente, se indica si un fichero es de código, de vectores de test, si es un paquete, o una selección de dispositivo.

Figura 5. Detalle de las ventanas de ficheros fuente y de procesos

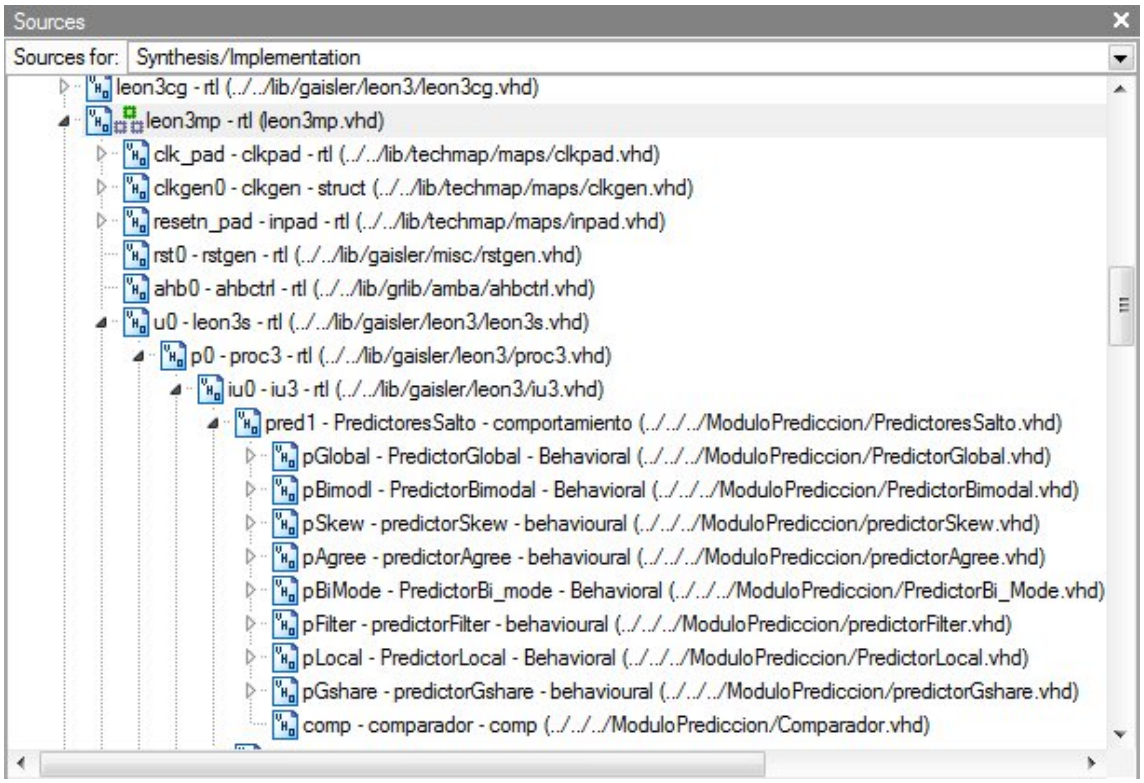
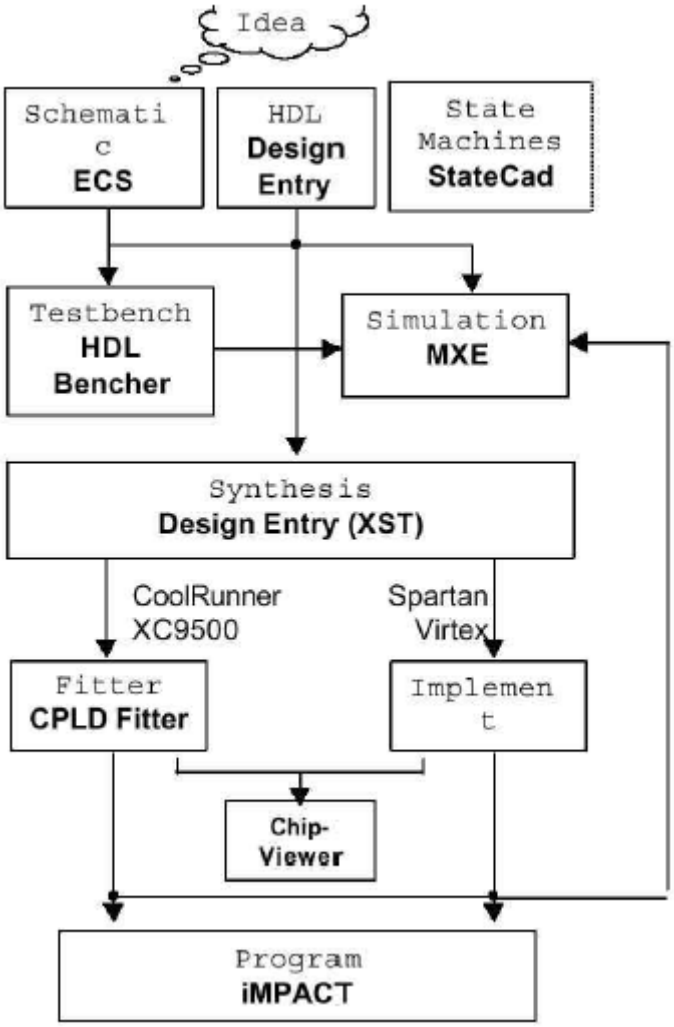


Figura 6. Proceso de diseño en Xilinx ISE



En concreto, la ventana de procesos incorpora todas las opciones necesarias para realizar todos los pasos de implementación de sistemas en lógica programable, incluyendo la edición y verificación. La figura 6 muestra el diagrama de flujo de diseño en Xilinx ISE, y la figura 7 muestra las diversas partes en que se divide la ventana de procesos dependiendo de la tarea a realizar.

Figura 7. División de tareas dentro de la ventana de procesos

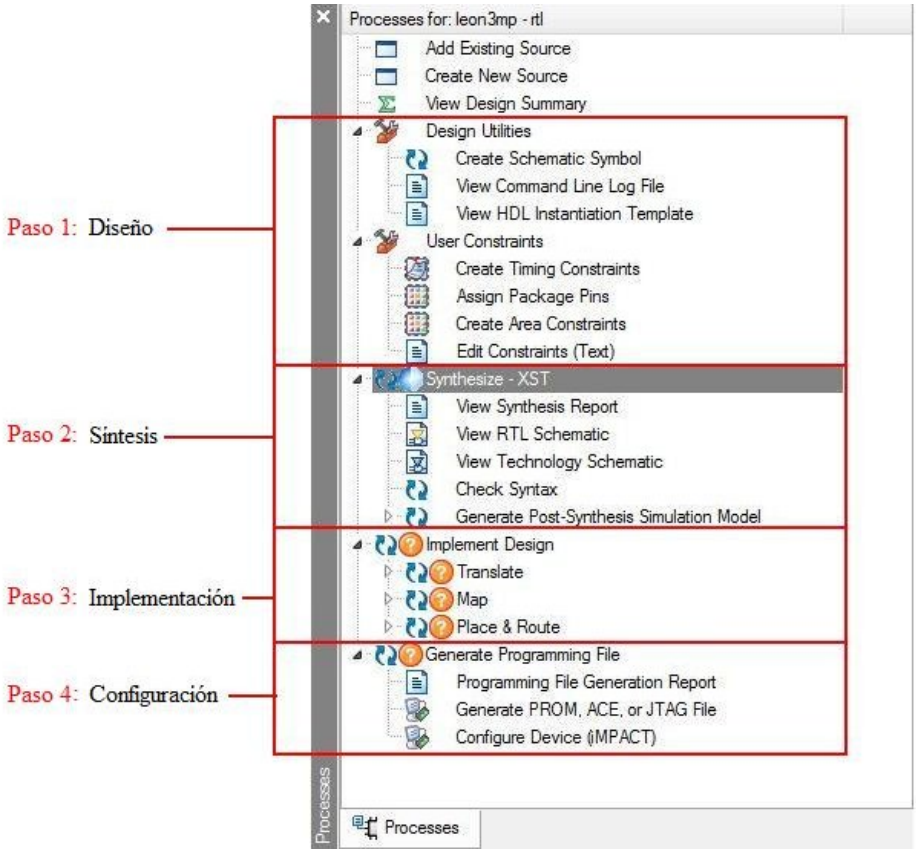
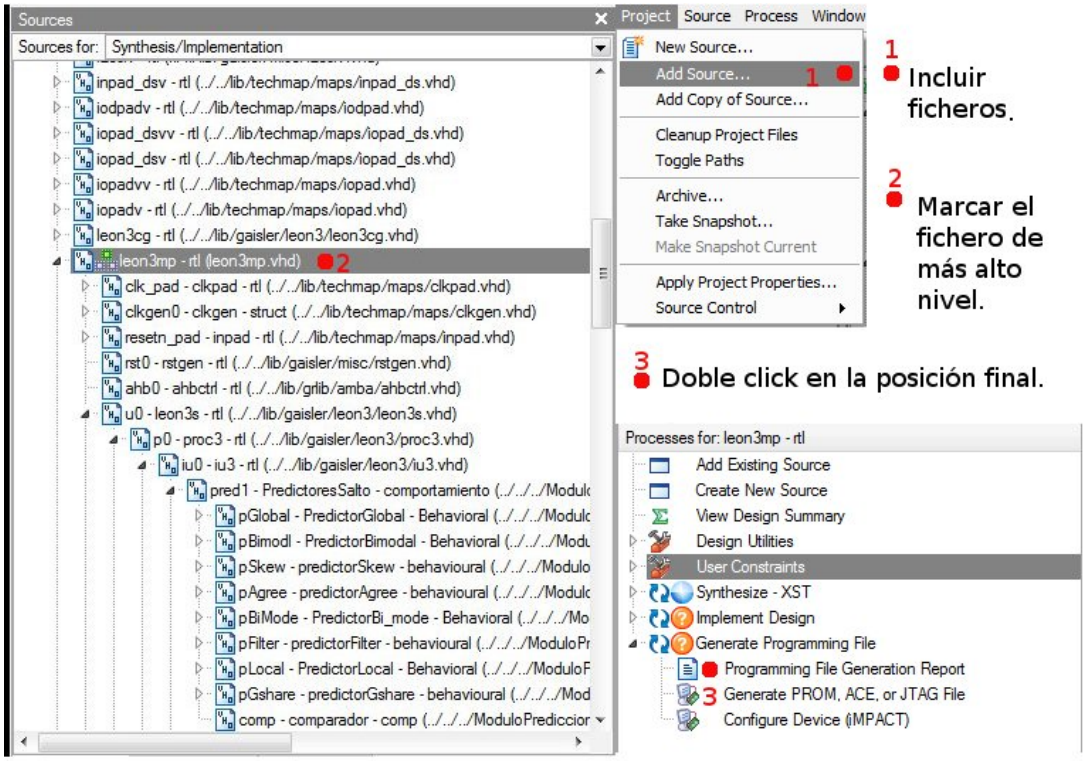


Figura 8. Proceso simplificado para el desarrollo de un diseño en Xilinx ISE



De manera resumida, el proceso de diseño resulta sencillo y se realiza en tres pasos, el primero consiste en añadir los ficheros fuente, en el segundo paso se selecciona el fichero de más alto nivel que se quiere implementar, y finalmente se hace doble clic sobre el último proceso al que se desea llegar, de este modo se ejecutarán todos los procesos intermedios necesarios para llegar al proceso seleccionado en último lugar (figura 6).

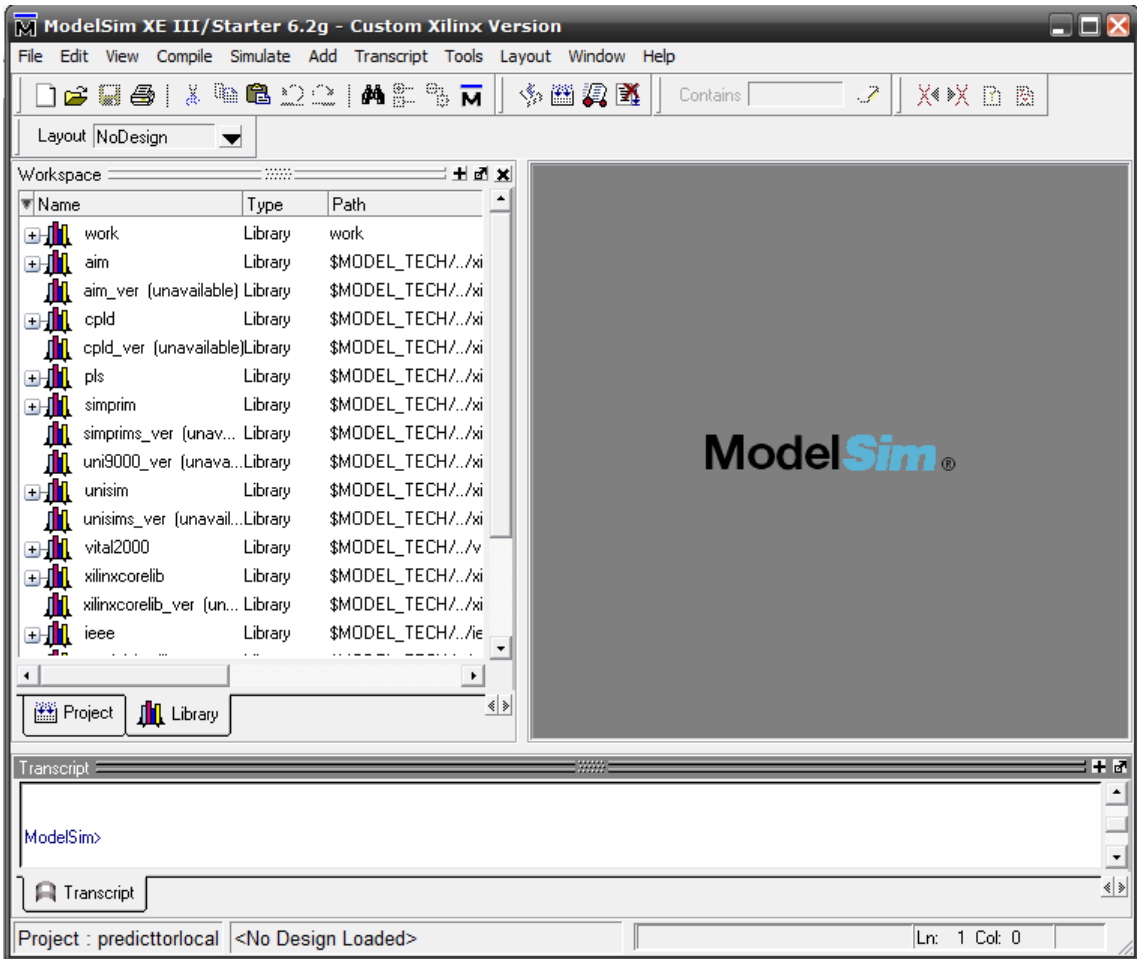
En el proyecto, este software se utilizó para la implementación en VHDL del gestor y para el testeo de los módulos básicos, con el propio simulador que incorpora ISE.

ModelSim 6.2g

Los bancos de prueba (o *testbenchs*) son una parte esencial del proceso de diseño, ya que permiten comprobar su correcto funcionamiento y ayudan en la automatización del proceso de verificación del diseño. Recoger estadísticas de la cobertura del código durante la simulación ayuda a asegurar la calidad y la minuciosidad de las pruebas.

El software ModelSim SE 6.2g utilizado en este proyecto, proporciona un entorno integrado de depuración (*Integrated Debug Environment*) que facilita el depurado eficiente de diseños basados en FPGAs, programados en cualquiera de los tres lenguajes siguientes: VHDL, Verilog y SystemC.

Figura 9. Ventana principal del simulador ModelSim SE 6.2g



Además de simular el funcionamiento del sistema diseñado, el ModelSim SE 6.2g permite visualizar las señales de cada puerto del bloque simulado y realizar las modificaciones necesarias del código en función del resultado obtenido.

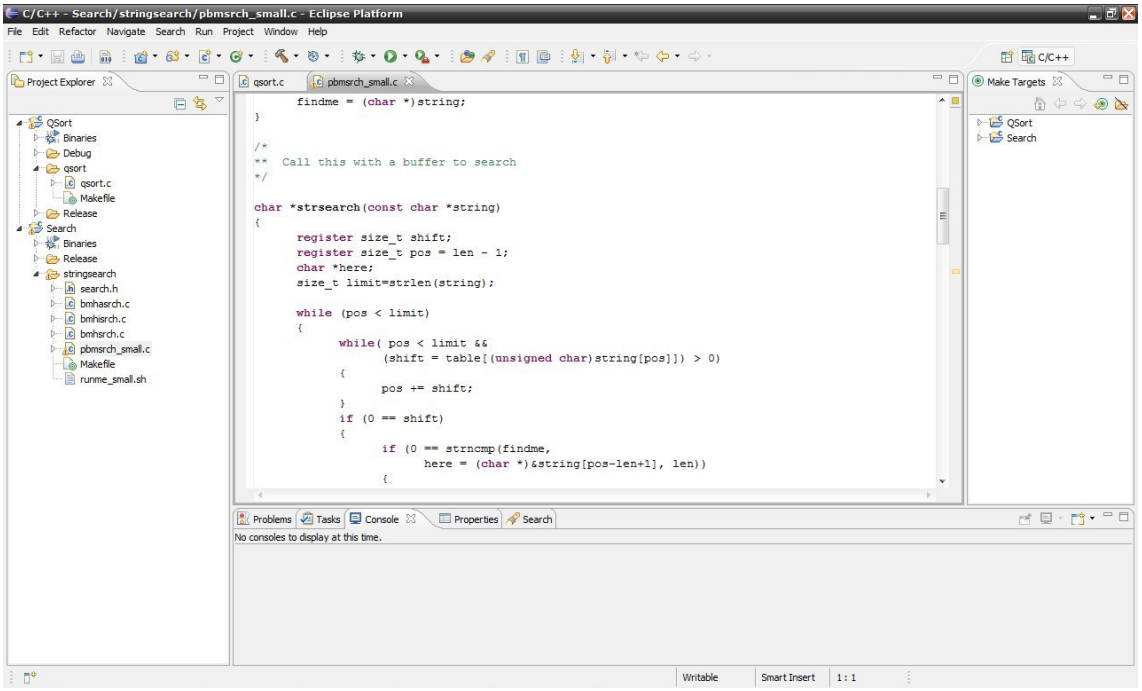
Partiendo del código proporcionado por el programa Xilinx ISE, la simulación se puede realizar a distintos niveles: desde el nivel más alto en el que se utilizan modelos “ideales” de los componentes hasta el nivel Post Place & Route donde cada componente ha sido mapeado a un elemento concreto de la FPGA y posteriormente han sido conectadas entre sí, de forma que la simulación se basa en modelos muy precisos de los componentes que incluso incluyen los retardos de las conexiones.

GRTools

El paquete de desarrollo software GRTools de Gaisler Research consiste en un archivo autoinstalable para Windows. En concreto contiene las siguientes herramientas:

- Compilador BCC.
    - Compilador para código C, C++ basado en arquitecturas SPARC v8.
- Compilador RCC, incluyendo RTERMS.
- LEON IDE, incluyendo Eclipse y CDT.
    - Entorno de desarrollo de aplicaciones software, basado en Eclipse, una herramienta de libre distribución de código java, que cuenta con numerosas extensiones.

Figura 10. Entorno de desarrollo C para SPARC v8

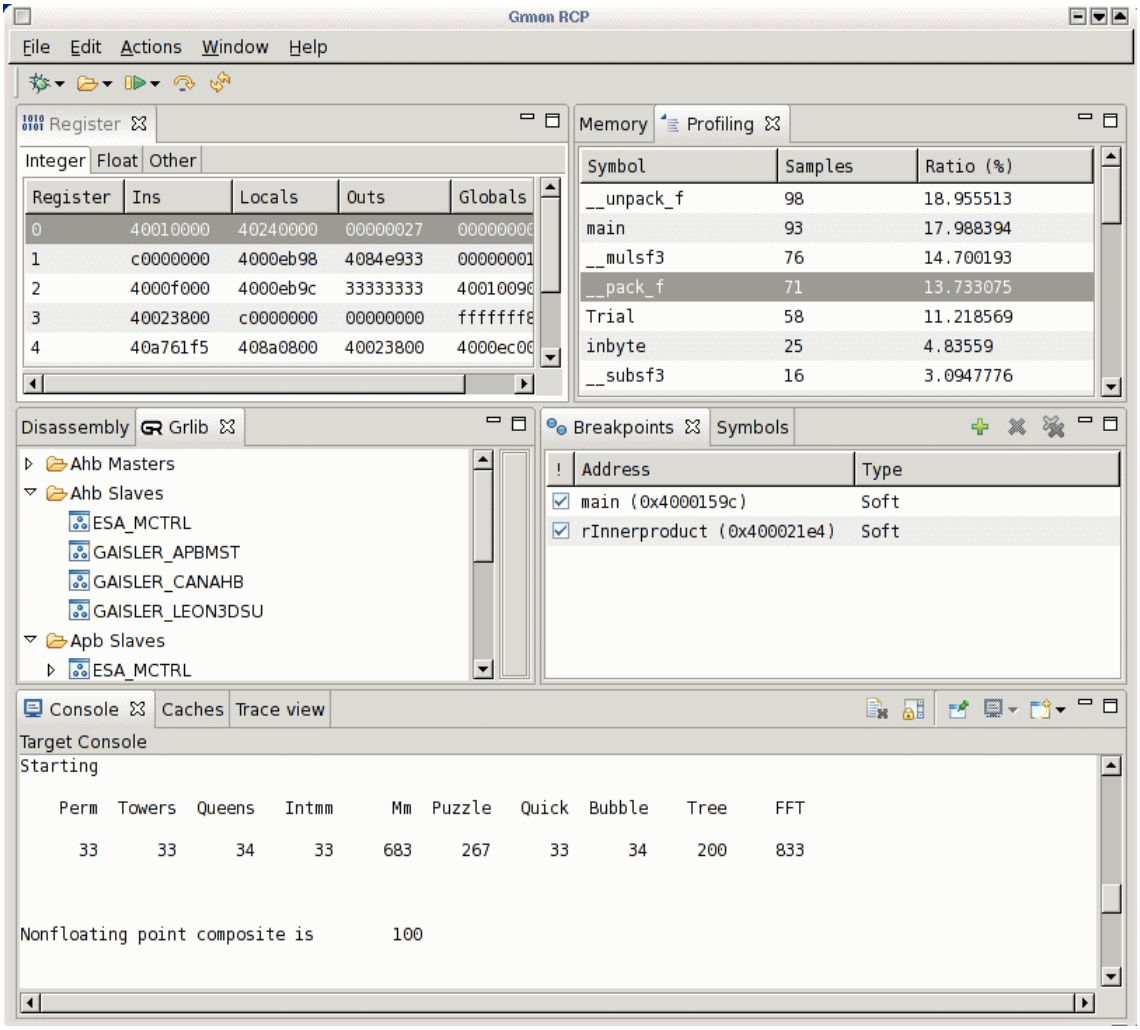


- GRMON.
    - Monitor para debug de procesadores Leon. Se comunica con la unidad de debug del Leon (DSU) y permite un debug no intrusivo del sistema objetivo completo. Provee las siguientes funciones:
        - Lectura/escritura de todos los registros y memoria.
        - Control del trace buffer.
        - Descarga y ejecución de aplicaciones.
        - Control de breakpoitns y watchpoints.

- Conexión remota al debugger GNU (gdb).
- Posibilidad de añadir comandos personalizados.
- Programación flash para flash proms de Intel y AMD.
- Soporte para la interfaz plug-and-play del Leon 3.
- Soporte para la integración en el entorno Leon IDE de Eclipse.
- Interfaces de debug por puerto serie, Ethernet, JTAG y USB.

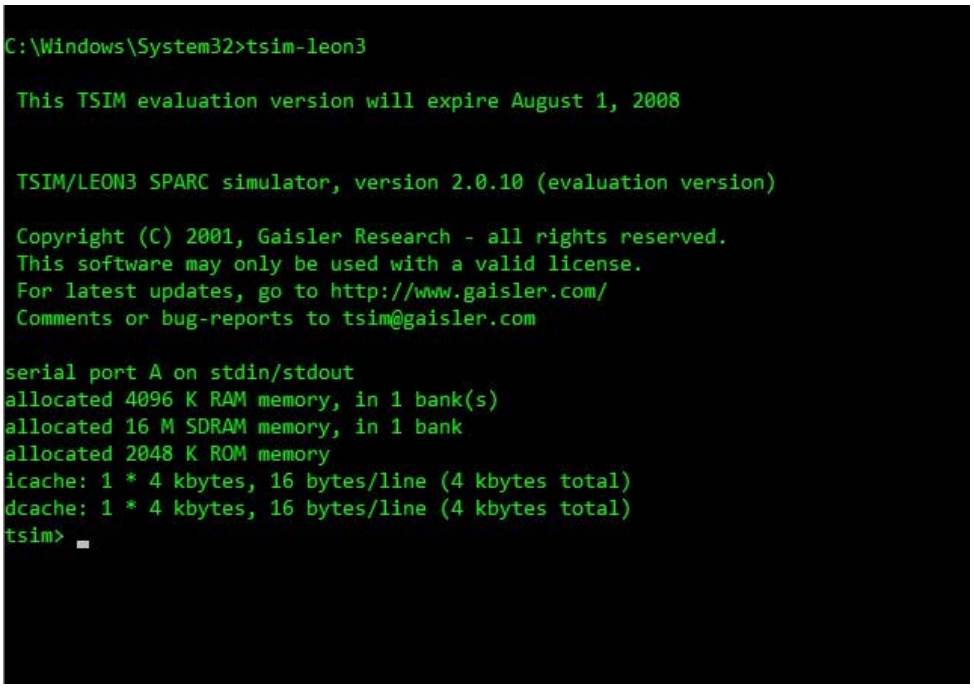
- GrmonRCP.
  - Interfaz grafica para el GRMON, en concreto se encuentra integrada bajo Eclipse.

Figura 11. GRMON RCP – Carga y debug de programas en el Leon 3



- TSIM-Leon 3. Simulador del Leon 3

Figura 12. Simulador TSIM



- Drivers HASP HL para GRMON y TSIM
- Herramientas de desarrollo:
  - MSYS.
  - MinGW.
  - MSYS DTK.
  - Autoconf.
  - AutoMake.

CygWin Bash Shell

CygWin es un emulador de entorno Linux para Windows. Consiste en dos partes:

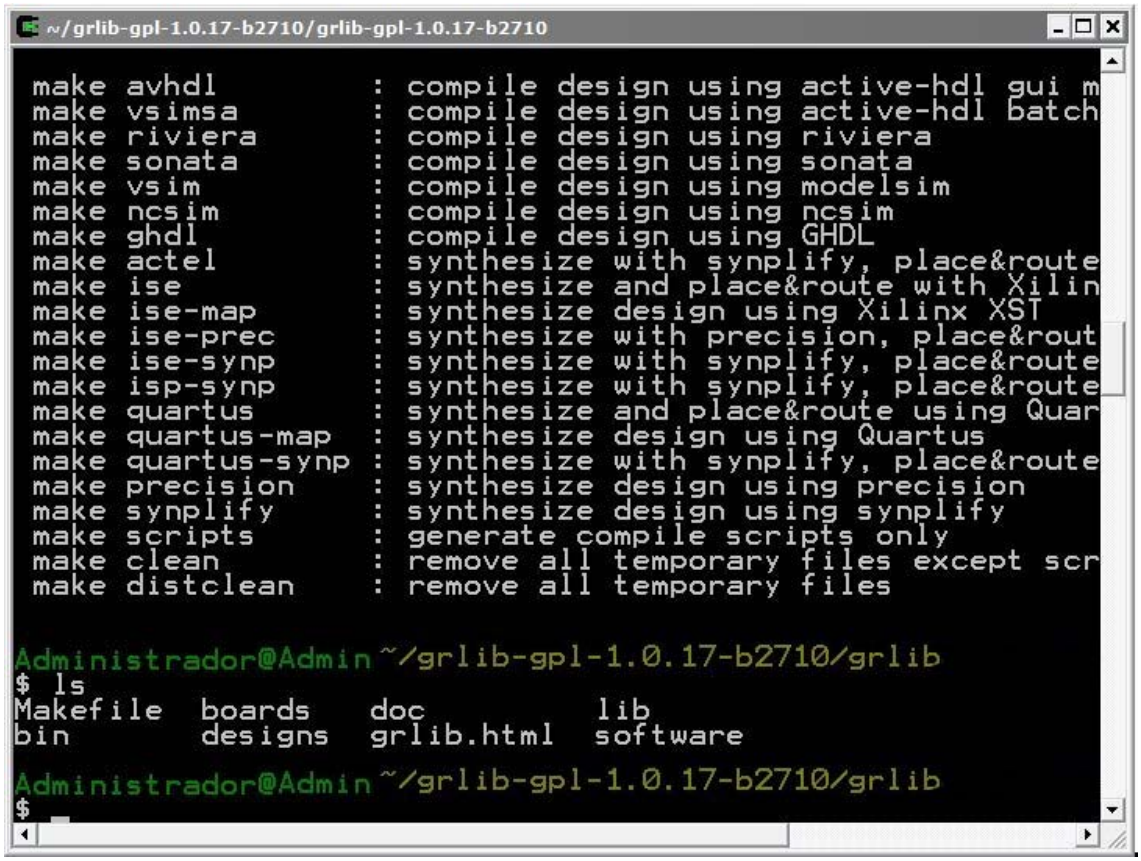
- Un DLL (cygwin1.dll) que actua como una capa de emulación API de Linux, otorgando funcionalidades sustanciales.
- Un conjunto de herramientas que dan una sensación de estar trabajando bajo Linux.

Cygwin funciona para cualquier Windows, tanto para ordenadores x86 de 32 bits como de 64, con la excepción de Windows Ce y Windows Mobile.

CygWin funciona como un terminal de Linux, permitiendo la ejecución de comandos sencillos nativos de Linux, en un entorno de desarrollo Windows. No permite la ejecución de aplicaciones nativas de Linux.

El programa en si tiene la gran ventaja de que al instalarlo permite seleccionar una serie de librerias disponibles dentro de un gran repositorio. La elección correcta de estas librerias es crucial para poder ejecutar correctamente ciertos programas, por lo que más adelante, en el capitulo “Preparación del entorno de desarrollo” haremos mencion de las utilizadas en nuestro caso.

Figura 13. Ventana de CygWin



Tortoise SVN – Cliente Subversion

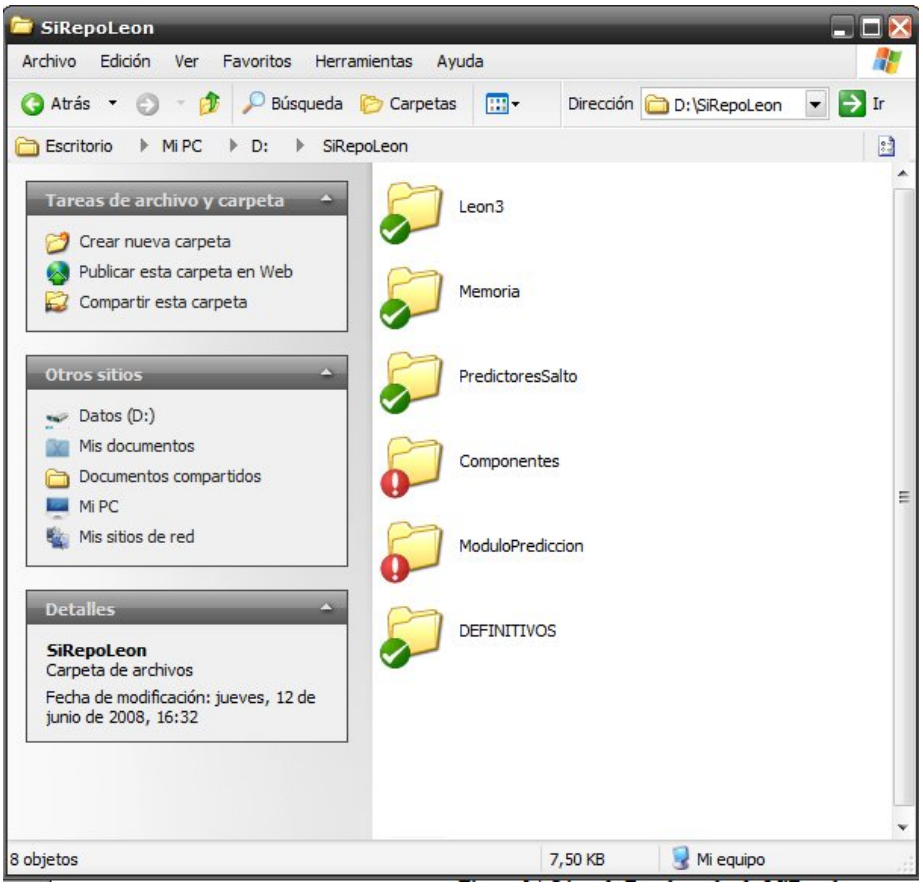
A la hora de escribir nuestro código, rápidamente se hizo patente la necesidad de mantener un sistema de control de versiones con nuestras modificaciones. Para ello, nos decantamos por Subversion, al ser una herramienta de libre distribución y código abierto. Una vez montado el servidor en si donde se alojarían estas versiones, nos decantamos por el cliente de SVN Tortoise. Este cliente, muy intuitivo y que se integra bajo Windows como una extensión del explorador, es también gratuito por lo que su elección fue bastante fácil.

Funciona de la siguiente manera:

- Fácil de usar
  - Todos los comandos se ejecutan directamente desde el explorador de Windows.
  - Solo se muestran los comandos que tienen sentido para el archivo o carpeta seleccionado. No se muestran nunca los comandos que no puedes usar en tu situación.
  - Posibilidad de ver el estado de los archivos directamente desde el explorador de Windows.
  - Gran cantidad de diálogos, tanto de información como de error, para dar mejor servicio al usuario.
  - Permite mover archivos arrastrándolos directamente desde el explorador.
- Se soportan todos los protocolos de Subversion.
  - http://
  - https://
  - svn://
  - svn+ssh://

  - file:///
  - Svn+XXX://
- Potente dialogo para confirmación (commit)
  - Corrector ortográfico integrado para los mensajes de log.
  - Función de auto-completar preparada para rutas y palabras claves de archivos modificados.
  - Formato de texto avanzado con caracteres especiales.
- Resumen gráfico
  - Posibilidad de crear un gráfico con todas las revisiones/commit. Esto permite una rápida y fácil visión del historial de trabajo.
  - Gráficos de estadísticas de los commits del proyecto.
  - Fácil comparación entre saltos o etiquetas.
- Herramientas extra
  - TortoiseMerge
    - Muestra los cambios hechos en tus archivos.
    - Ayudante para la resolución de conflictos.
  - TortoiseBlame: Para ver los mensajes de log de cada archivo, línea a línea
  - TortoiseDiff: compara en paralelo dos archivos y resalta las diferencias entre cada uno.
- Totalmente traducido.

Figura 14. Ejemplo de uso de Tortoise



Benchmarks para Sistemas Empotrados MiBench

A la hora de mostrar los resultados de nuestro proyecto, se hizo necesaria la búsqueda de algunas aplicaciones útiles para la medida de resultados. Sin embargo, al no poseer el Leon 3 de un modulo propio para tratar archivos, así como de la carencia nativa de un sistema

efectivo de entrada y salida (tanto de video, como para control de teclado, ratón y audio) la elección de estos programas de benchmark no resultaba fácil.

El uso entonces de los benchmark incluidos en el paquete de MiBench [18] nos pareció acertado. Este paquete, gratuito y representativo de sistemas empotrados comerciales está desarrollado por la Universidad de Michigan, se encuentra disponible para su descarga en la pagina web http://www.eecs.umich.edu/mibench/, contiene las siguientes aplicaciones:

Tabla 1. Lista de benchmarks pertenecientes a las suite MiBench

Auto./Industrial	Consumer	Office	Network	Security	Telecomm.
basicmath	jpeg	ghostscript	dijkstra	blowfish enc.	CRC32
bitcount	lame	ispell	patricia	blowfish dec.	FFT
qsort	mad	rsynth	(CRC32)	pgp sign	IFFT
susan (edges)	tiff2bw	sphinx	(sha)	pgp verify	ADPCM enc.
susan (corners)	tiff2rgba	stringsearch	(blowfish)	rijndael enc.	ADPCM dec.
susan (smoothing)	tiffdither			rijndael dec.	GSM enc.
	tiffmedian			sha	GSM dec.
	typeset				

Como se puede observar, se encuentran divididos en 6 secciones, claramente diferenciadas. Sin embargo, debido a las limitaciones del procesador, no era posible compilar cada benchmark y hacerlo funcionar para la arquitectura SPARC v8. Esto hizo por ejemplo que las secciones de “Consumer” y “Telecomm” no estén tratadas en nuestros benchmarks, ya que era imposible sin un sistema operativo nativo tener un sistema de ficheros para convertir algo a jpeg, por ejemplo, o de igual manera, mandar un email encriptado sin tener una configuración básica de red.

Gracias a la aplicación de Eclipse mencionada anteriormente, y a que disponíamos del código fuente al ser estos benchmarks gratuitos, compilamos algunos especialmente descriptivos (ver apéndice “*Herramientas y Método de compilación de aplicaciones para el Leon 3*”), siempre teniendo en cuenta que si bien en algunos originalmente se pedía la carga por teclado o por un fichero externo algunos datos, nuestros benchmarks disponen de estos datos precargados. A continuación exponemos los usados para nuestro proyecto:

- BasicMath:

  El test “basic math” realiza cálculos matemáticos simples que normalmente no tienen un soporte hardware dedicado dentro de los procesadores empotrados. Por ejemplo, la resolución de funciones cúbicas, raíces cuadradas enteras o conversión de ángulos de grados a radianes, funciones todas ellas necesarias para calcular una velocidad de un automóvil u otros valores vectoriales. El valor de entrada son una serie de datos contantes.

- BitCount:

  El algoritmo de cuenta de bit, BitCount, comprueba las capacidades de tratamiento de datos de un procesador contando el número de bits en un array de enteros. Para esto, usa 5 métodos:

  1. Un contador básico.
  2. Un contador optimizado de un bit por iteración.
  3. Un contador recursivo de un bit por partes.
  4. Un contador de un bit no recursivo por partes usando una tabla de búsqueda.

5. Un contador no recursivo de un bit por bytes usando una tabla de búsqueda y bits de desplazamiento y cuenta.

El valor de la entrada es un array de enteros con igual numero de 0's y 1's.

- QuickSort:

  La prueba de Qsort ordena un array de enteros en orden ascendente usando el algoritmo, de sobra conocido, quick sort. El ordenamiento de información es importante para los sistemas para que se puedan establecer, por ejemplo, un sistema de prioridades, una salida por pantalla más fácilmente interpretable y en definitiva para que el tiempo de ejecución de un programa se vea reducido. El valor de entrada consiste en mil enteros ordenados en orden completamente descendente.

- StringSearch:

  Este benchmark busca unas palabras dadas en frases usando un algoritmo de comparación intensivo.

- Dijkstra:

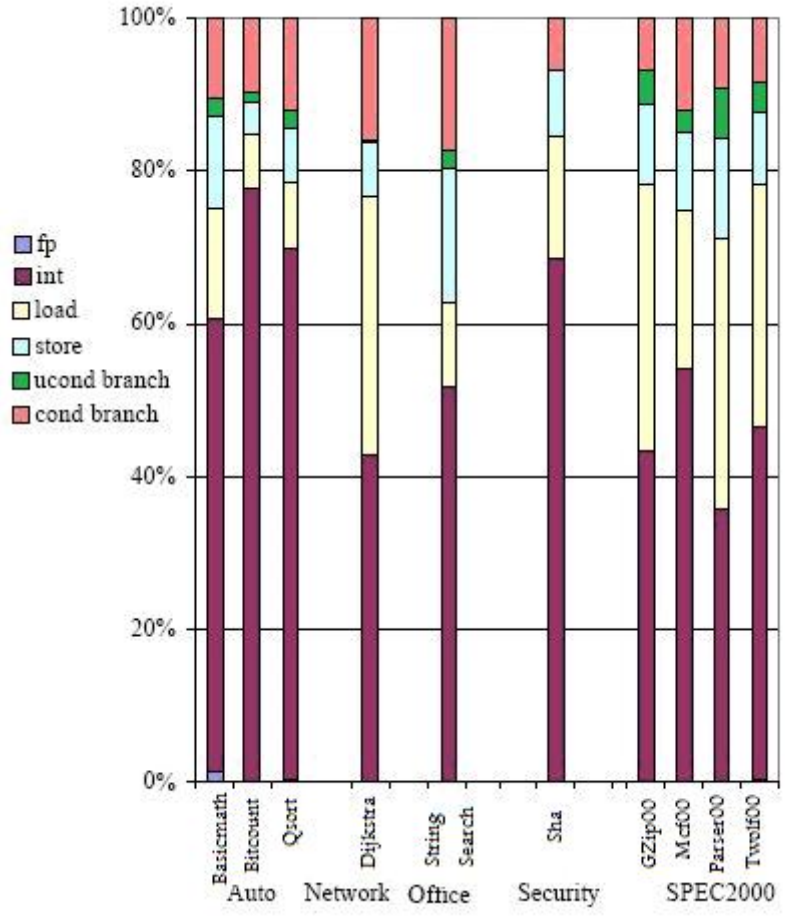
  El benchmark Dijkstra crea un grafo, representado en una matriz de adyacencia, para después calcular la ruta mas corta entre cada par de nodos usando repetidas llamadas al algoritmo de Dijkstra. Este famoso algoritmo soluciona el problema del camino mas corto y lo completa en orden de $O(n^2)$.

- Sha:

  SHA es un algoritmo de seguridad hash que genera un mensaje de 160 bits para una determinada entrada. Se suele utilizar para garantizar el intercambio seguro de claves criptográficas y para generar firmas digitales. Se usa incluso en las conocidas funciones hash MD4 y MD5.

Por su particular interés, reproducimos a continuación una grafica donde se observan para cada benchmark el porcentaje de tipo de instrucciones que contienen, frente a la suite SPEC2000. En ella, podemos observar por ejemplo que en los benchmarks elegidos la carga de instrucciones enteras es muy superior a las de SPEC2000. Así mismo, se realizan menos operaciones de carga de datos. Por ultimo, observamos también como las instrucciones de salto condicional son algo mayores dentro de la suite MiBench, característica particularmente importante de cara a la medición de resultados para nuestro proyecto por motivos obvios.

Figura 15. Porcentaje de tipos de instrucciones para MiBench



Por ultimo, decidimos usar un benchmark que el responsable del Leon 3, Gaisler Research, proveía en su página web. Este es el Drhystone, conocido por formar parte a su vez de SPECInt, ampliamos un poco la historia y las características de este benchmark:

- Dhrystone fue creado con la intención de medir el rendimiento en ordenadores, no en sistemas empotrados. Se creó en 1984, por lo que no es de extrañar que su creador, el Doctor Reinhold P. Weicker, por aquél entonces en Siemens AG, se centrase en el rendimiento de enteros. La versión actual del Dhrystone, la versión 2.1 se creó en 1988 y permanece inalterada desde entonces. Weicker hizo Dhrystone como modelo de lo que debería ser visto como una aplicación “típica” con mezcla de operaciones matemáticas y otros tipos.

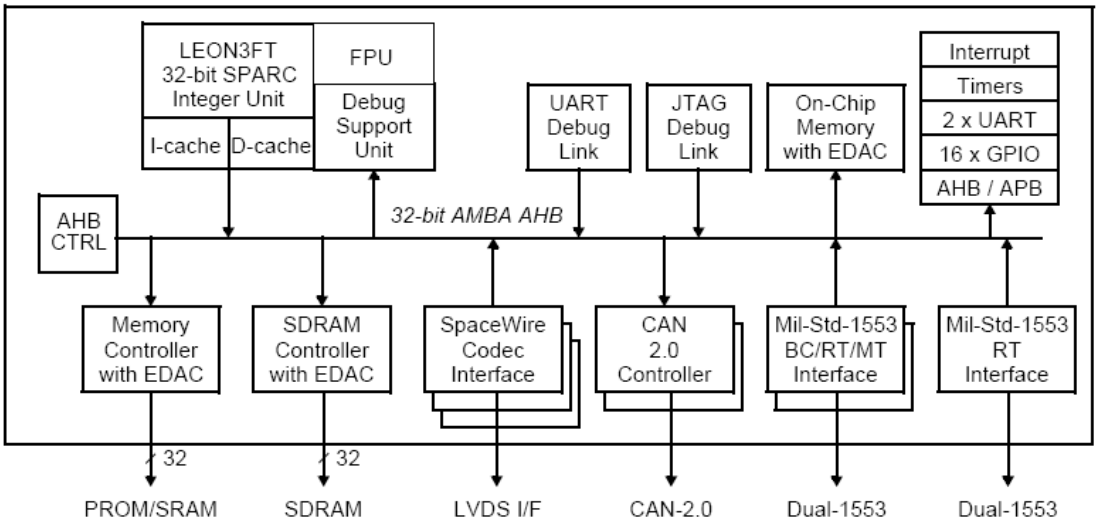
  Sin embargo el calculo de rendimiento de enteros predomina, con poco o ningún calculo en punto flotante. Esta aplicación puede usarse dentro de pequeñas cantidades de memoria, de manera que con el paso del tiempo este benchmark ha dejado de usarse en ordenadores completos y sobre todo, a consecuencia del gran avance de las arquitecturas de computadores, ha acabado siendo usado en su mayoría para los sistemas que al principio excluía, los sistemas empotrados.

DESARROLLO DEL PROYECTO

VISIÓN GENERAL DEL PROYECTO

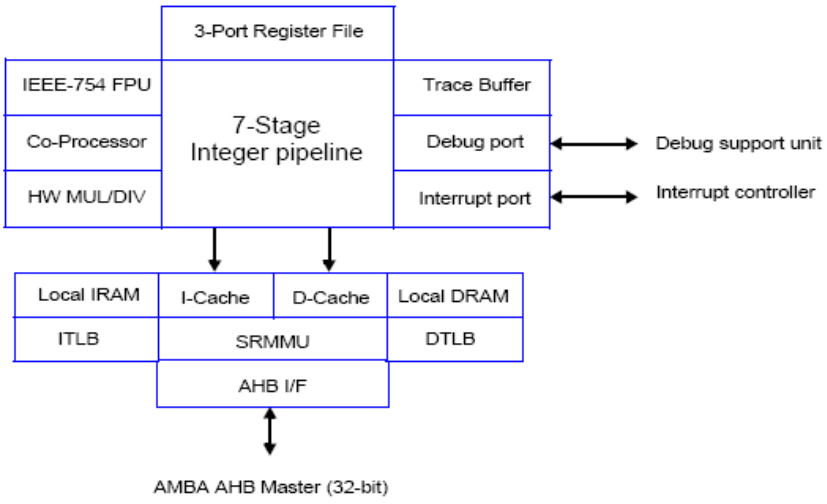
Nuestro proyecto consiste en una modificación al procesador GPL IEEE-1754 Leon 3. Este parte de la arquitectura SPARC v8 y no cuenta con ningún tipo de predicción de saltos.

Figura 16. Diagrama arquitectónico de un sistema que usa Leon 3



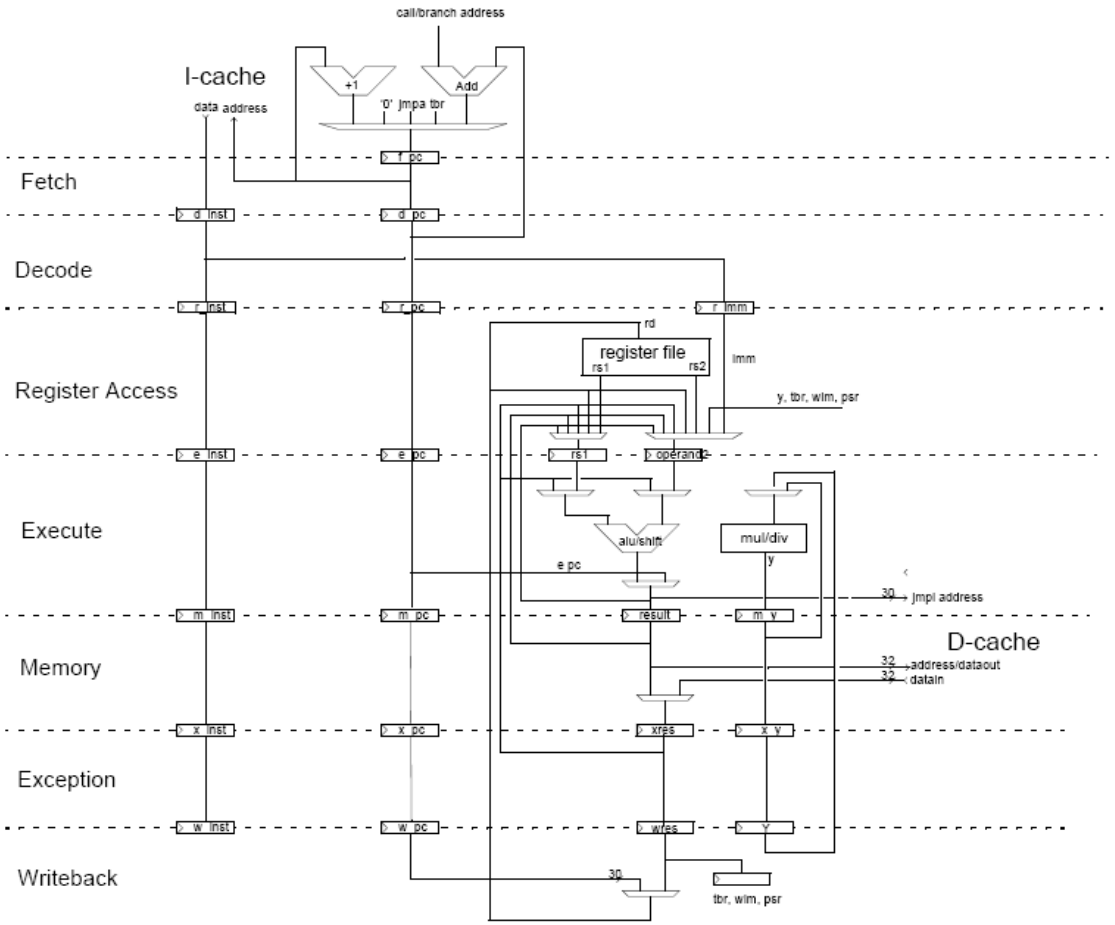
Por motivos de capacidad principalmente, aunque también por problemas con la sincronización en el caso de la memoria SRAM externa, en nuestro caso no se encuentran todos los módulos que están disponibles. Sin embargo, para extraer estadísticas de una forma sencilla incluimos un modulo de pantalla especifico para la muestra por pantalla de nuestros resultados, haciendo uso de las capacidades de la FPGA, pero que funciona de manera independiente al procesador, y que está adecuado a las necesidades de nuestro proyecto.

Figura 17. Diagrama de bloques del procesador Leon 3



Dentro del procesador SPARC, en la capa más superior, hicimos una conexión directa de este modulo VGA con nuestro comparador y generador de estadísticas. A su vez, una vez adentrándonos en el procesador, llegamos hasta el modulo iu3.vhd, donde se encuentra la ruta de datos del Leon 3. La siguiente figura muestra esta ruta de datos:

Figura 18. Pipeline (7 etapas) de la unidad de enteros del Leon 3



Nuestras modificaciones se centraron por tanto en este modulo. Siendo aquí donde incorporamos nuestro modulo PredictoresSalto.vhd. Dentro de este modulo se encuentran los siete predictores desarrollados en el proyecto, a saber:

- Global
- Bimodal
- Local
- GShare
- Filter
- Bi-Mode
- Skew

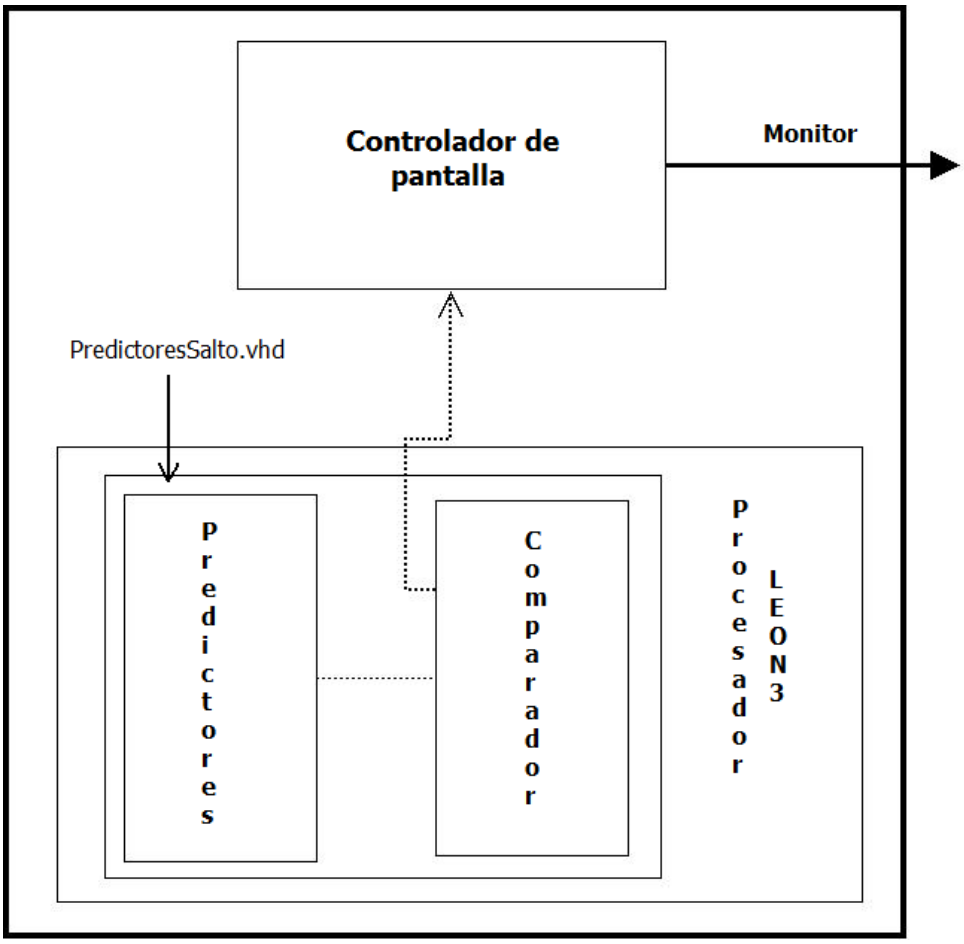
Este modulo se encuentra al mismo del pipeline, pues hace uso de las señales necesarias para identificar una señal de salto condicional (no contamos los saltos incondicionales pues no tendría sentido su predicción), así como el PC para cada salto predicho así como, lógicamente, si el salto se llego a tomar o no para el calculo de las estadísticas. Es importante destacar por tanto el funcionamiento del Leon 3 frente a las instrucciones de salto. Cuando un programa al compilar detecta una instrucción de salto condicional, genera una instrucción de comparación previa a la identificación del salto. Esto supone que el salto en si, la instrucción BICC como por ejemplo un 'BEQ', permanezca en la fase Decode hasta que se conoce el resultado de la comparación. Una vez hecho esto, el salto lee las señales de control de la ALU de enteros y toma una decisión respecto al siguiente PC. La siguiente tabla expresa un ejemplo correcto de este comportamiento:

Tabla 2. Ejemplo de ejecución de una instrucción de salto condicional

Instrucción												
CMP	F	D	R	X	M	E	W					
BICC		F	D	D	D	R	X	M	E	W		
***			F	F	F	D	R	X	M	E	W	
PC correcto						F	D	R	X	M	E	W

En el ejemplo hemos considerado una instrucción inmediatamente después a la de salto condicional. Esta instrucción se genera en tiempo de compilación, de manera que se rellena como un hueco de salto (Branch Delay Slot), que puede ser una instrucción útil, o una instrucción de relleno tipo 'NOP'. Es precisamente aquí donde los predictores de salto adquieren su importancia, puesto que en vez de dos o más ciclos de espera, podría reducirse a uno o normalmente ninguno, por lo que la ganancia en rendimiento es indudable. Las siguientes imágenes muestran un esquema resumido de las conexiones dentro del procesador, omitiendo control de periféricos y otros módulos que no intervienen en el objetivo del proyecto:

Figura 19. Resumen esquemático del proyecto



Por ultimo, dentro del modulo de predictores, se encuentra la instancia (o *port map*, como se designa en lenguaje VHDL) a cada predictor especifico. Tras esto, una vez sintetizado, traducido a la FPGA considerada y cargado en la misma, ejecutamos varios benchmarks comerciales preparados para arquitecturas empotradas, destinados a medir la tasa de aciertos de cada predictor. Estos predictores son a su vez variables en su tamaño, de manera que se posibilita la opción de medir el rendimiento de cada uno según su coste en hardware (puesto que a más tamaño, mayor precio).

PREPARACIÓN DEL ENTORNO DE DESARROLLO

El paquete GRLib proporcionado por Gaisler Research está preparado para que pueda ser sintetizado, simulado y ejecutado en múltiples plataformas. En primera instancia, será necesario configurar las características y opciones internas del procesador, para lo cual se requerirá el uso de un Bash como es Cygwin.

Cygwin

En las opciones de instalación, comenzaremos seleccionando las opciones por defecto para el tipo básico de fichero. A continuación, aparecerá una ventana preguntando por las extensiones a instalar al programa. En ese momento, deberemos seleccionar para ser instaladas tres librerías básicas para la síntesis del procesador Leon 3. Dichas librerías son gcc-core, make y tcltk.

Figura 20. Librería gcc-core

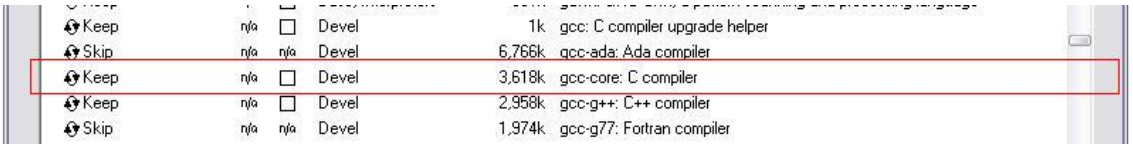


Figura 21. Librería make

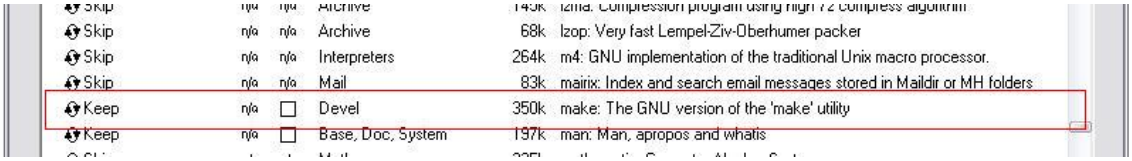
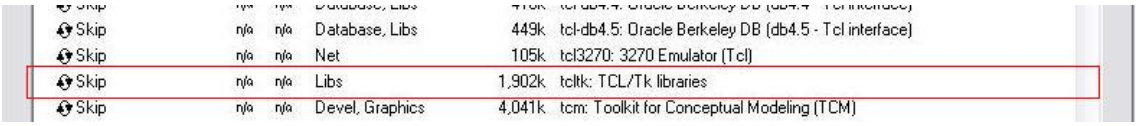


Figura 22. Librería TCL/Tk



Dependiendo de la versión de Cygwin que se instale, en ocasiones los pasos siguientes de preparación del sistema pueden producir errores o funcionar incorrectamente. Esto puede ser debido a un problema con la función make, o con la librería TCL/Tk.

Xconfig

Ésta es la herramienta principal de personalización del procesador. Esta interfaz gráfica permite modificar todos los parámetros del Leon 3, tales como el número de procesadores conectados al bus de sistema, la velocidad del reloj principal que gobierna el funcionamiento del sistema, los módulos periféricos del procesador (memoria RAM, unidad de punto flotante, sistema de caché o debug support unit, entre otros), así como sus componentes internos (número de ventanas de registro o la latencia de determinadas unidades de trabajo).

Figura 23. Vista principal de la utilidad Xconfig del Leon 3

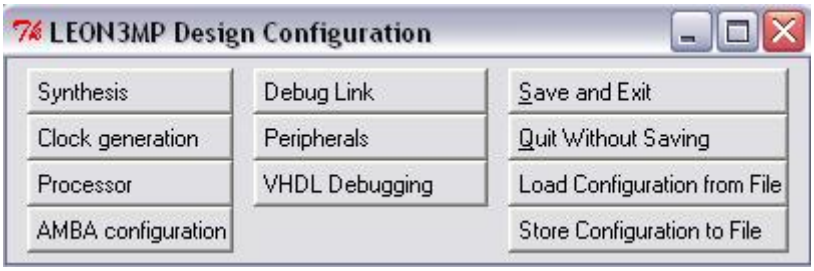
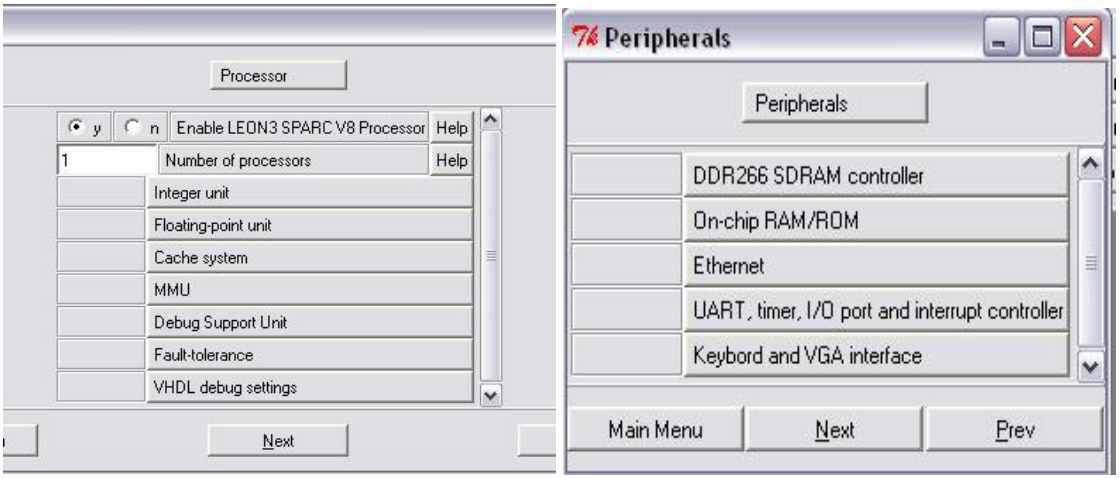


Figura 24. Propiedades varias dentro de Xconfig



Debido a las propiedades de nuestro proyecto, para el correcto funcionamiento del mismo se deberán establecer los siguientes parámetros:

- Processor → Floating-Point Unit → Enable FPU: n
- Peripherals → DDR266 SDRAM Controller → Enable DDR266 SDRAM controller: n
- Peripherals → On-chip RAM/ROM → On-chip AHB RAM: y
- Peripherals → On-chip RAM/ROM → AHB RAM size: 64
- Peripherals → On-chip RAM/ROM → RAM start adress: 400
- Peripherals → Ethernet → Gaisler Research […]: n
- Peripherals → Keyboard and VGA interface → Keyboard/Mouse: n
- Peripherals → Keyboard and VGA interface → Text-based VGA interface: n
- Peripherals → Keyboard and VGA interface → SVGA graphical frame buffer: n

Una vez hecho esto, salimos de la interfaz salvando las modificaciones. Esta operación habrá sido correctamente realizada si al final obtenemos un mensaje de confirmación por parte de xconfig con la leyenda “config.vhd created”. El fichero indicado en ese mensaje se modifica en este momento, y no necesitaremos volver a editarlo en lo que resta de realización de proyecto. En caso de querer modificar algún parámetro de configuración del procesador sin necesidad de utilizar el interfaz proporcionado por xconfig, bastará con hacerlo directamente en este archivo.

Ahora será necesario acceder al código fuente del procesador para realizar algunas modificaciones. Al deshabilitar la FPU, la RAM externa y la interfaz de teclado y ratón, existen líneas del fichero de mapeo Leon3mp.ucf y de Leon3mp.xcf que deberán ser eliminadas. Para simplificar esta tarea, adjuntamos dichos ficheros convenientemente editados en el CD que acompaña a esta memoria.

Una vez modificados los ficheros pertinentes, introducimos en la consola (o *Bash*) el comando *make ise* para comenzar una síntesis cuyo resultado producirá un proyecto completo para ser cargado desde Xilinx ISE 8.1 bajo el nombre Leon3mp.npl. Abriendo este proyecto con Xilinx ISE 9.2i, se pedirá que el proyecto sea migrado a esta nueva versión.

DESCRIPCION DEL SISTEMA

Predictores de Salto

Existen dos tipos de estrategias de predicción, dependiendo de cuando se realice la predicción, por un lado tenemos las técnicas de predicción estática, realizadas en tiempos de compilación, y consistentes en mecanismos tan simples, como por ejemplo suponer que todos los saltos se tomarán, o que solo se tomaran los saltos hacia atrás; este tipo de predicciones destacan por un menos coste en HW si las comparamos con las predicciones dinámicas, a coste de tener menos precisión y un aumento en el tamaño del ejecutable que puede llegar incluso a un 30%.

Por otra parte las técnicas de predicción dinámica, ofertan unas predicciones más certeras y precisas, a cambio de un mayor coste en HW, su funcionamiento se basa en información obtenida en tiempo de ejecución.

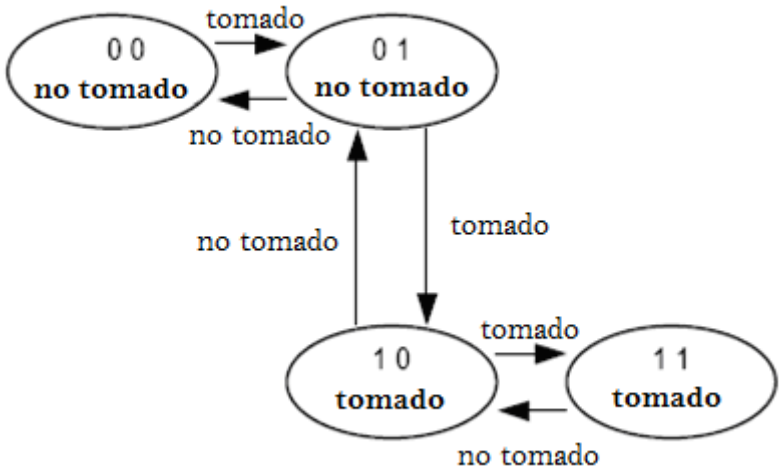
Los siete predictores de salto (todos dinámicos) implementados en el presente proyecto parten de ideas comunes, y se puede observar como progresivamente cada predictor ha refinado los conceptos básicos, en busca de un mayor índice de aciertos. A continuación exponemos para cada uno, tanto el fundamento teórico en el que esta basado, como el diseño que hemos implementado.

En el CD adjunto a la presente memoria adjuntamos el código VHDL de la totalidad de los predictores aquí expuestos. Hemos optado por una implementación parametrizada de modo que el tamaño del predictor pueda ser fijado de una manera sencilla, simplemente definiendo la cantidad de bits de entrada que tiene cada tabla dentro del diseño de los predictores.

La idea clave, en torno a la cual gira todo lo demás, consiste en que los saltos no tienen un comportamiento tan aleatorio como podría parecer en un principio, muchos saltos presentan uno tipo fuertemente marcado, según el cual tenderán irremediablemente hacia el ser tomados o por caso opuesto el no ser tomados. Las estrategias de diseño de Predictores de Salto toman esta última afirmación como núcleo sobre el cual basar su diseño.

El siguiente esquema concreta la idea anteriormente mostrada:

Figura 25. Esquema básico de un predictor de dos bits de estado



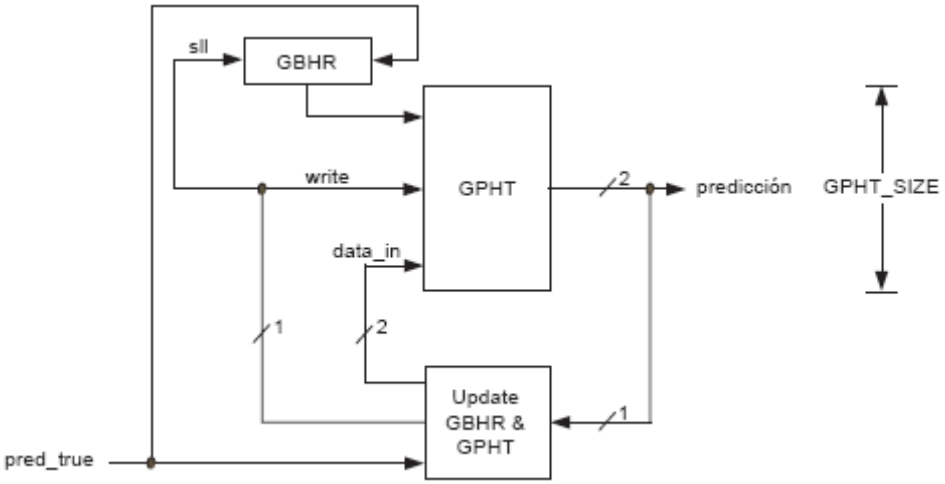
La unidad atómica de cualquier predictor es un contador saturado módulo 4; si lo representamos en formato de Maquina Finita de Estados, nos encontramos con que los posibles estados por los cuales se puede catalogar a una instrucción de salto en concreto son: Fuertemente No Tomado, No Tomado, Tomado y Fuertemente Tomado. El significado semántico es obvio, sin embargo el motivo de utilizar 4 estados en vez de 2, es para evitar que un comportamiento aislado, cambie la salida del sistema, esto es: un salto que tienda a ser tomado, es posible que en ocasiones concretas no se tome, sin embargo esto ultimo, por ser una excepción, no debe condicionar el comportamiento global de la predicción.

La actualización del contador, se hará en función de si dada una instrucción de salto, el salto se llega a tomar o no, estas decisiones irán condicionando el estado del contador, de modo que si una instrucción de salto se toma en la mayoría de los casos, el estado de su contador asociado será Tomado o Fuertemente tomado.

Por ultimo, al final de la explicación de cada predictor implementado incluimos una tabla comparativa de ocupación de la FPGA según el número de entradas a cada uno, siendo la ultima cantidad la previa al umbral en cuanto a la limitación de tamaño. Así mismo, destacamos el hecho de que para todos los predictores y versiones de los mismos, el tiempo de ciclo no se ve reducido o afectado, puesto que toda la evaluación y actualización de los elementos necesarios para su funcionamiento, nunca llegan a ser mayores que los ciclos del propio procesador Leon 3.

Global

Figura 26. Esquema del predictor Global



En el Predictor Global [20] utilizamos para cada salto la información de sus últimas N ejecuciones, otro posible modo de enfocar el estudio del comportamiento de un salto es tener en consideración los últimos saltos tomados por un programa; para ello bastará con un registro de desplazamiento (GBHR), donde se va almacenando el resultado de los últimos M saltos realizados en el programa, el contenido de ese registro, servirá para direccionar una entrada de la BHT (aquí llamada GPHT (Global Predicción History Table), para mayor contenido semántico).

El predictor de saltos global se beneficia de 2 tipos diferentes de comportamiento, por un lado la dirección tomada por el salto actual (es decir se ha tomado o no) puede depender en gran medida de lo ocurrido en saltos anteriores, véanse por ejemplo el ejemplo siguiente:

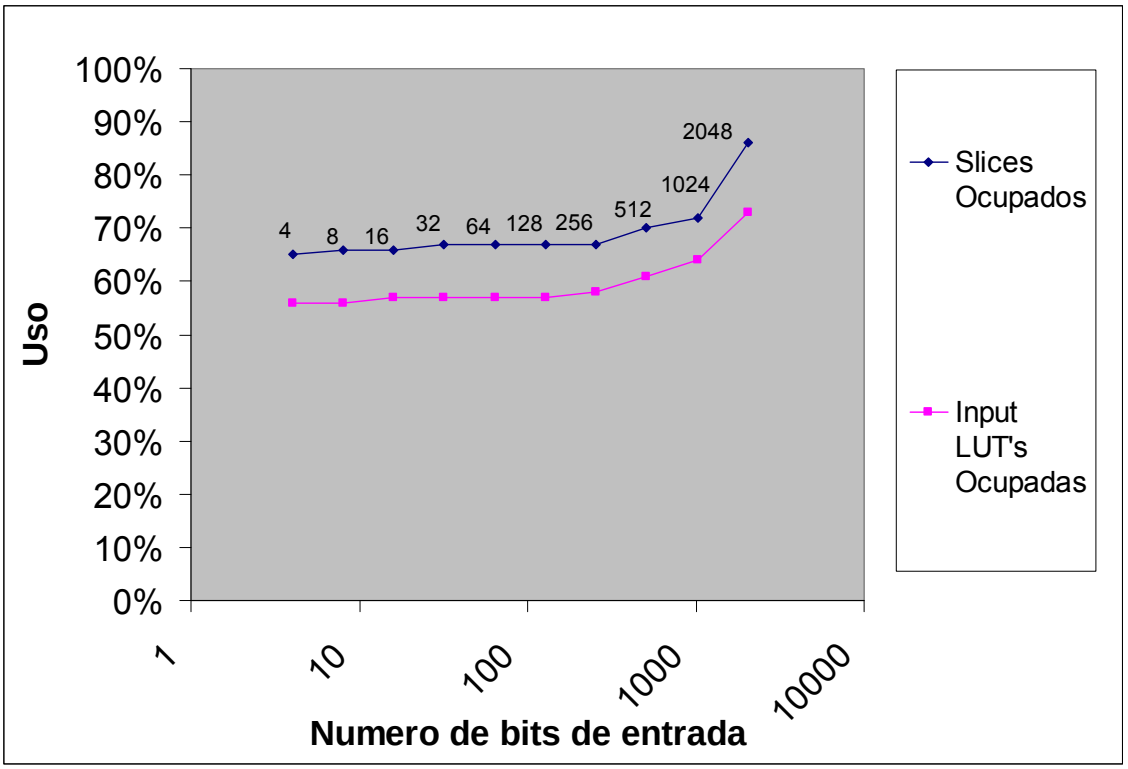
[salto 1]If (x >=1) then ….

[salto 2] If (x < 1) then …

Sabiendo el resultado del primer salto, podemos predecir sin miedo a equivocarnos el resultado del segundo.

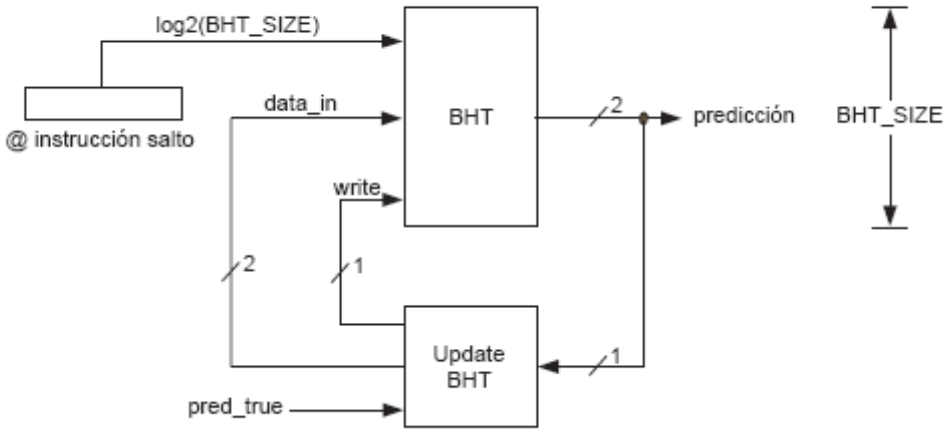
El segundo modo en que la predicción de saltos global puede ser útil es mediante la duplicación del comportamiento de la predicción de los saltos locales. Esto puede ocurrir cuando la historia global incluya toda la historia local necesaria para hacer una predicción exacta.

Gráfica 1. Porcentaje de uso de la FPGA para el predictor Global



Bimodal

Figura 27. Esquema del predictor Bimodal



Si nos limitados a utilizar un único contador modulo 4, para todas las instrucciones de salto, nos encontraremos con que diferentes saltos con comportamientos diferentes actuando sobre un mismo contador (concepto denominado *Aliasing*) podrían llegar a adulterar los resultados; el concepto es que se debería usar un contador por instrucción de salto, para asegurarnos que no se mezclen comportamientos opuestos en un mismo contador.

Por desgracia, debido a cuestiones de espacio, no suele ser posible esta ultima afirmación, pero si se puede intentar que si no es factible el tener un contador por instrucción de salto, si podemos intentar que el numero de instrucciones de salto que comparten el mismo contador sea el mínimo, de dicha idea surge la idea del Predictor Bimodal [2] [4].

La BHT (Branch History Table) es una tabla con un numero definido de contadores de 4 estados (que funcionan como se indica en la figura 25), cada uno de los cuales sigue el comportamiento descrito en el párrafo anterior, el modo en el cual las diferentes instrucciones de salto son asignadas a una entrada concreta de la BHT (es decir a un contador concreto), es mediante el uso de los N bits menos significativos del contador de programa (PC). A mayor capacidad de la BHT, mas bits del PC harán falta para direccionar la entrada en concreto, factor inversamente proporcional al numero de instrucciones de salto que compartirán contador.

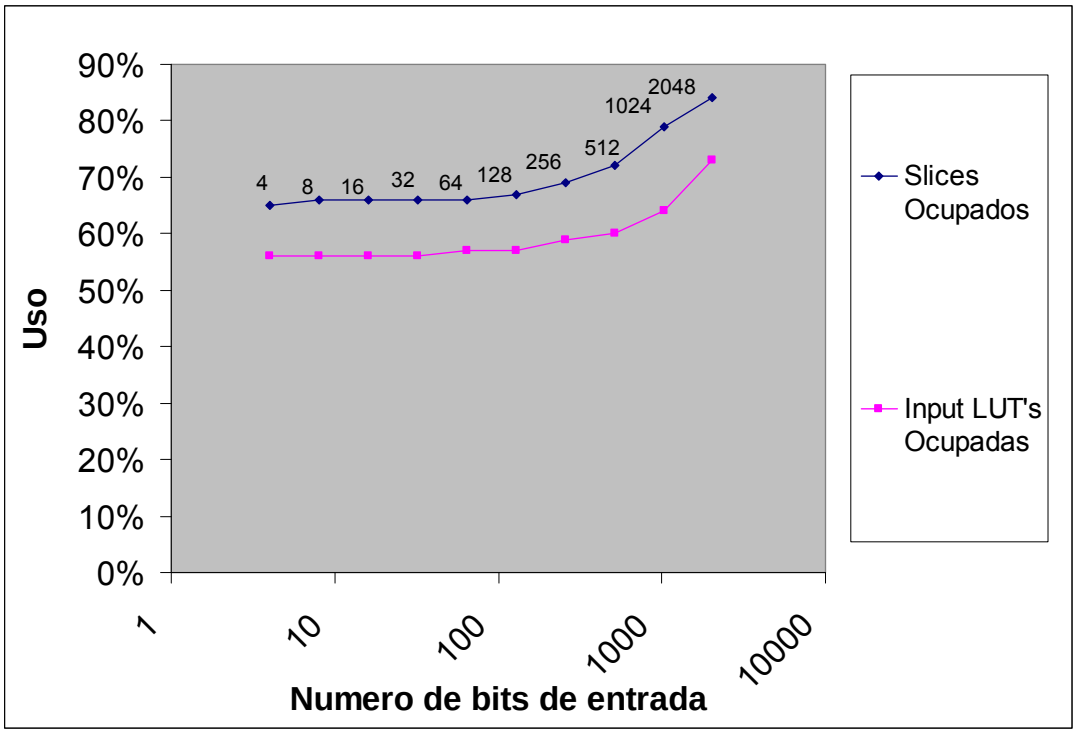
Los pasos en secuencia que sigue este predictor son los siguientes:

- Ante una instrucción de salto, se utilizan los N bits menos significativos del PC para direccionar una entrada de la BHT.

- El contador alojado en la entrada de la BHT direccionado por los bits menos significativos del PC, tendrá uno de los 4 posibles valores antes descritos, en caso de que dicho estado sea Tomado o Fuertemente Tomado la predicción dada será la de tomar el salto, en caso contrario el predictor recomendara no tomar el salto.

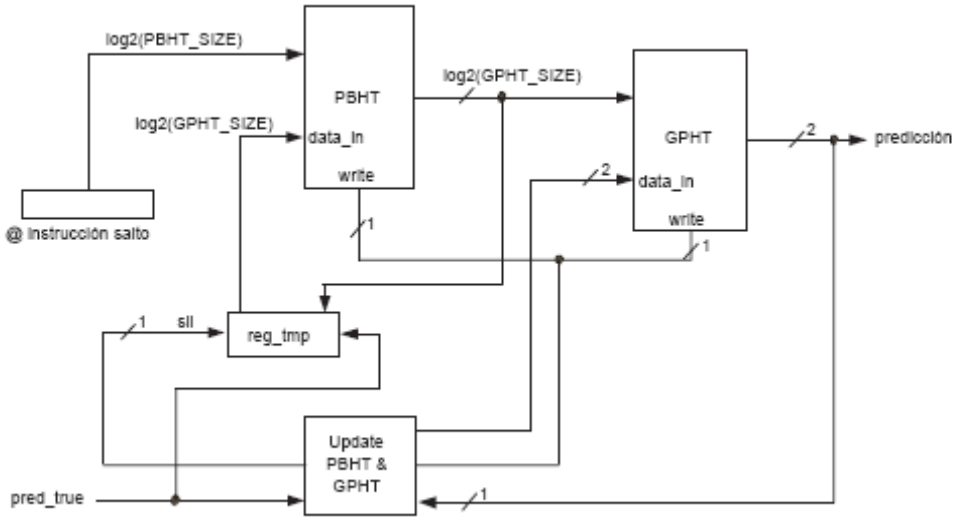
- Una vez se sepa el sentido real del salto, se actualizará la entrada correspondiente de la BHT, teniendo en cuenta que por las características de esta, un cambio de comportamiento aislado no repercutirá en la decisión global del contador.

Gráfica 2. Porcentaje de uso de la FPGA para el predictor Bimodal



Local

Figura 28. Esquema del predictor Local



Muchos de los saltos de un programa ejecutan patrones repetitivos, posiblemente el ejemplo más clarificador sea el de los bucles FOR, sea el siguiente bucle:

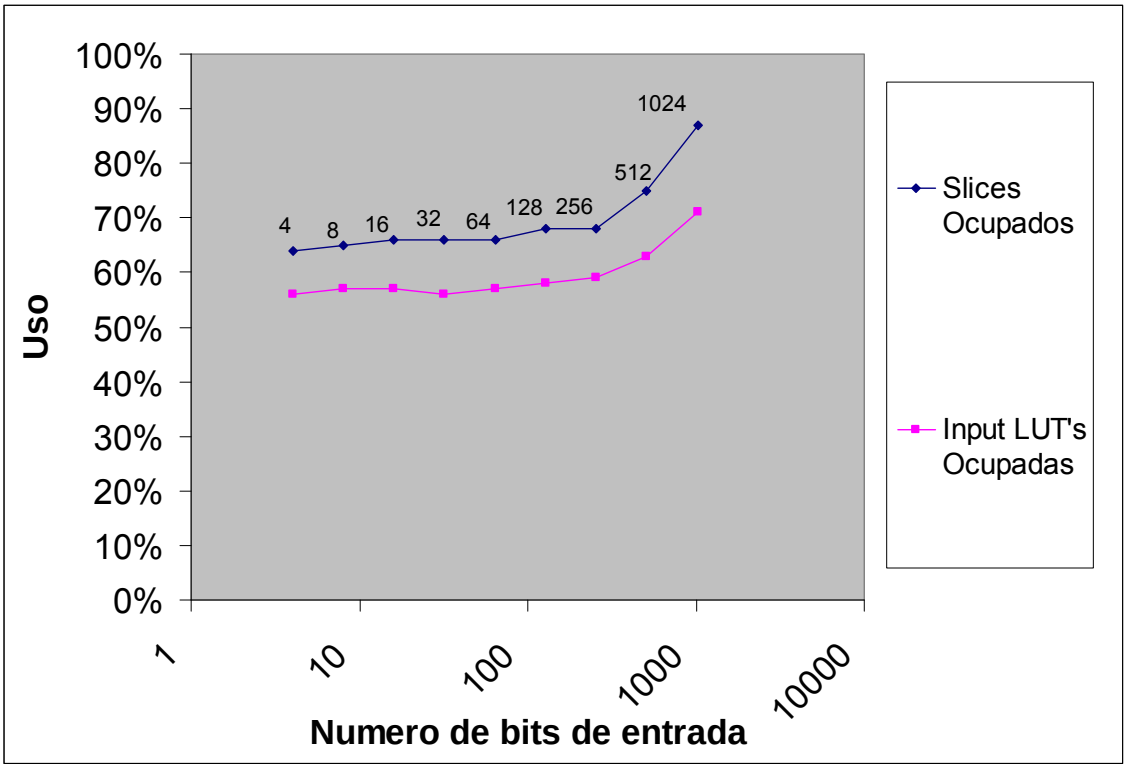
$$for\ (i=1;\ i<=4;\ i++)$$

Si ese bucle se ejecuta N veces, podemos concluir que la historia local para ese salto será $(1110)^n$, es decir sabiendo el resultado de las 3 ultimas ejecuciones de ese salto, podríamos predecir sin margen de error la 4º. Es decir: nos ayudamos de la historia local de ese salto (sus últimas ejecuciones) para predecir la siguiente [4].

Para almacenar la historia local de cada salto, usaremos un registro de desplazamiento, el registro de desplazamiento de cada salto será almacenado en una tabla (PBHT) y servirá para direccionar una determinada entrada en una segunda tabla que no será sino una BHT idéntica a la usada en el Predictor Bimodal.

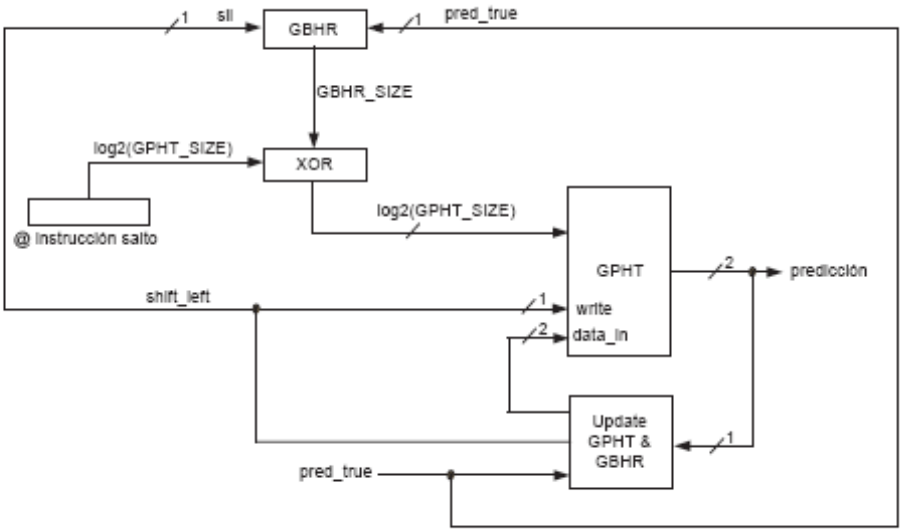
Al igual que con el Predictor Bimodal, la entrada de la PBHT es seleccionada por los bits menos significativos del PC, surgiendo el mismo problema encontrado en el Predictor Bimodal, un tamaño pequeño de la PBHT hará que se mezclen diversas historias locales, disminuyendo el porcentaje de acierto.

Gráfica 3. Porcentaje de uso de la FPGA para el predictor Local



GShare

Figura 29. Esquema del predictor GShare



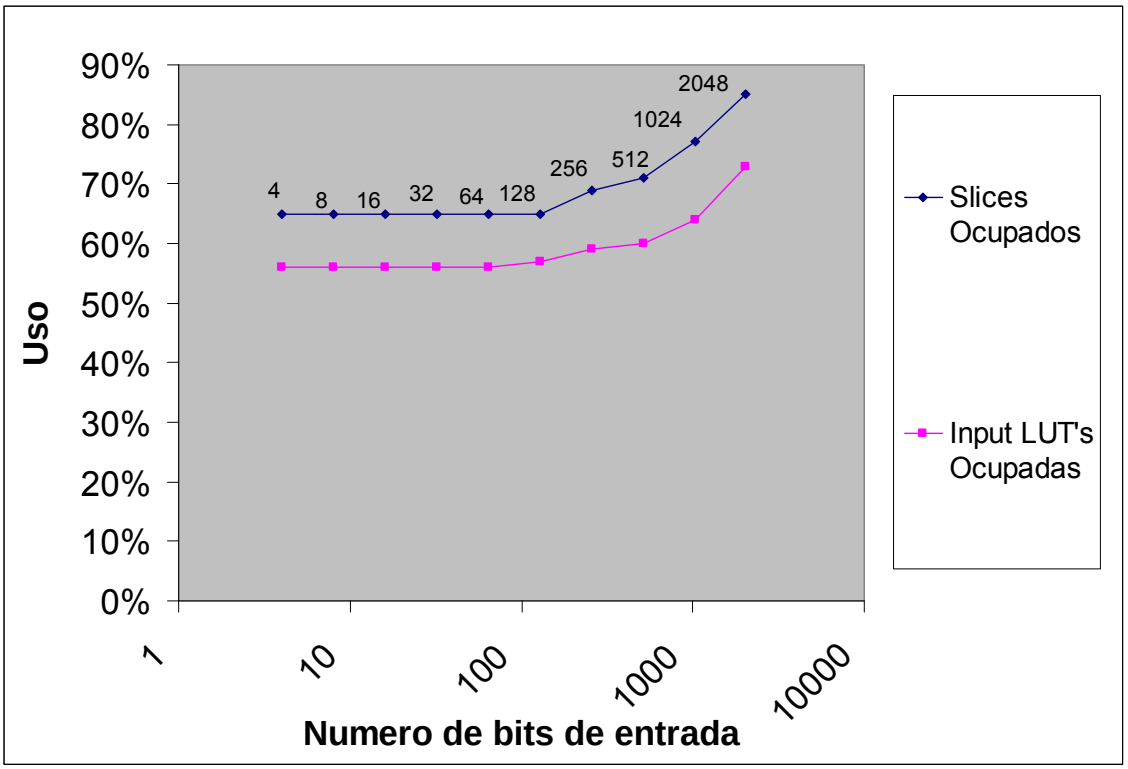
La Predicción Global tiene el inconveniente de que es menos eficiente a la hora de identificar el salto actual que cuando utilizamos la propia dirección del salto; estudios realizados muestran que se obtiene una mayor tasa de acierto cuando se utiliza conjuntamente la dirección de la instrucción y el registro de historia.

Las primeras aproximaciones abogaban por concatenar los M bits menos significativos de la dirección de la instrucción, con los N bits del registro de historia, obteniendo así la

dirección a leer de la BHT. Esta primera aproximación ofrece resultados ligeramente superiores a los obtenidos con técnicas de Predicción Locales y Globales, lo cual teniendo en cuenta que sólo estamos usando 1 tabla, en vez de las 2 que usa el Predictor Local, es una mejora significativa.

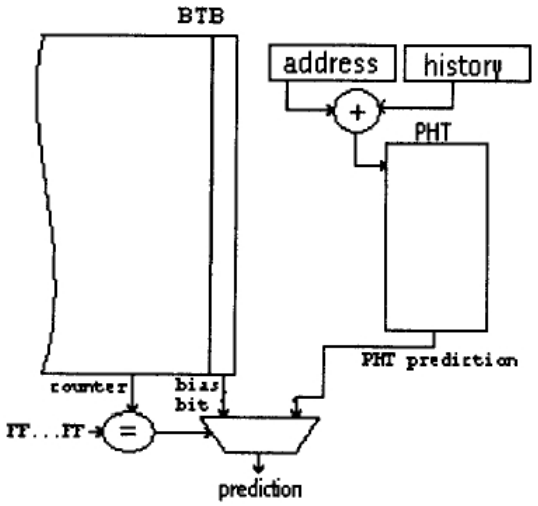
Posteriores estudios descubrieron que se podía obtener mas información (poder distinguir mas saltos diferentes), si en vez de concatenar los bits menos significativos de la dirección de la instrucción con los del registro de historia, haciendo una XOR de ambos datos. Como consecuencia de esta última afirmación tenemos el Predictor GShare [4].

Gráfica 4. Porcentaje de uso de la FPGA para el predictor GShare



Filter

Figura 30. Esquema del predictor Filter



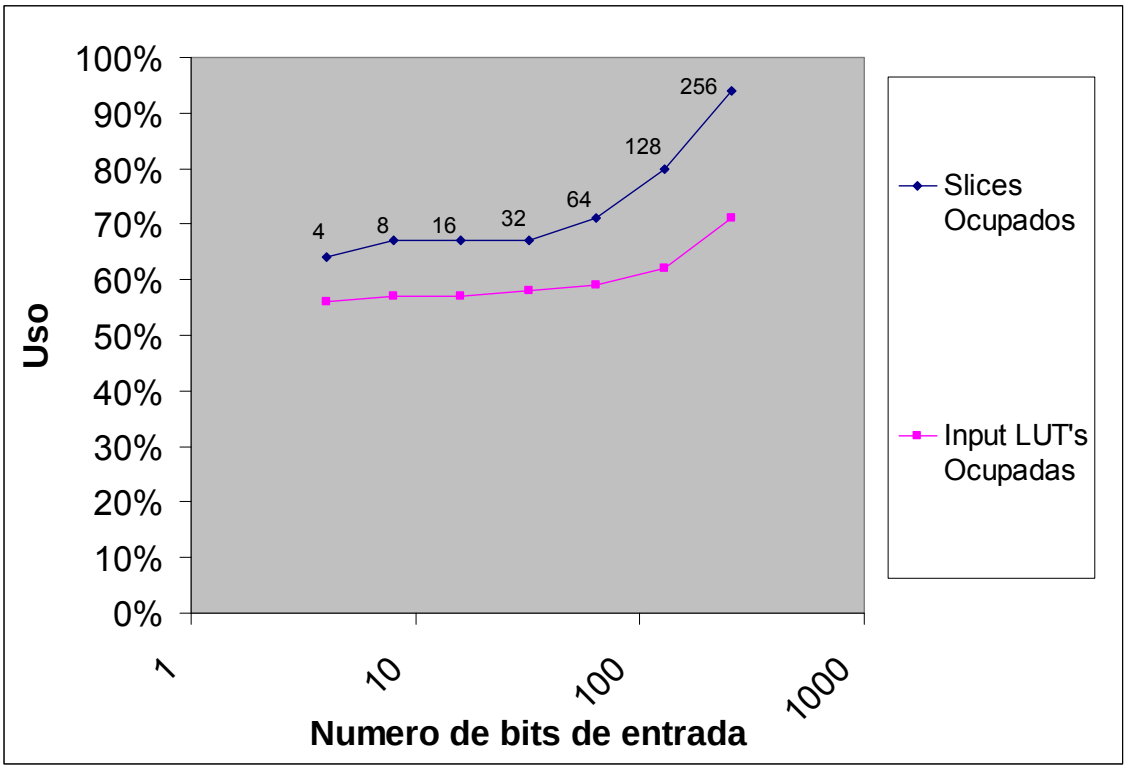
El objetivo principal de este esquema de predicción [7] es disminuir la cantidad de información redundante en las tablas del predictor. Aprovecha la idea de que saltos con tendencias muy fuertes pueden ser predichos con una tasa de acierto muy elevada con un bit únicamente.

Para ello, introduce un contador saturado en la BTB para cada una de las entradas. Mientras el contador almacena un valor distinto del máximo, cada salto se predice a través de la información almacenada en su entrada correspondiente de la BHT. Sin embargo, si el contador ha llegado a su valor máximo, se considera que se trata de un salto de tendencia fuerte y el dato usado para su evaluación será el proporcionado por la BTB.

Al ser introducido un salto en esta nueva BTB modificada, en el bit de tendencia se almacena la dirección del salto en esa primera aparición. Cuando el mismo salto vuelva a ejecutarse, si el sentido del salto coincide con la tendencia para esa entrada, el contador se incrementará en una unidad. En cambio, de no haber acertado, el contador se reseteará y el bit de tendencia cambiará su sentido.

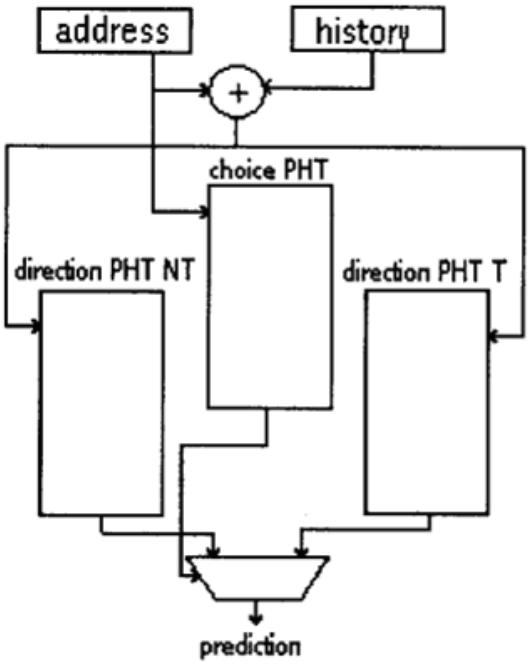
Estudios realizados han demostrado, además, que el valor óptimo de inicialización de dicho contador es a su valor máximo, de modo que el mecanismo de filtrado comience a funcionar desde el primer momento. Si el salto no muestra una tendencia fuerte, el contador se reseteará pronto y el bit de tendencia cambiará. Por el contrario, de inicializar el contador a cero, aunque el salto mostrase una tendencia fuerte, de haber sido el contador inicializado a cero, el sistema tardaría tiempo en reflejar esa tendencia, y por tanto, desaprovecharía las ventajas del sistema de filtrado.

Gráfica 5. Porcentaje de uso de la FPGA para el predictor Filter



Bi-Mode

Figura 31. Esquema del predictor Bi-Mode



El predictor Bi-Mode [10] tiene como objetivo el de paliar los efectos del Aliasing negativo; para eso divide la BHT en varias partes:

- BHT de Elección: Una BHT, con una ligera modificación en su método de escritura, cuya misión es elegir para cada caso, cual de las BHTs de Dirección dará la predicción; al igual que en casos anteriores está indexada por los bits menos significativos de la dirección de la instrucción.

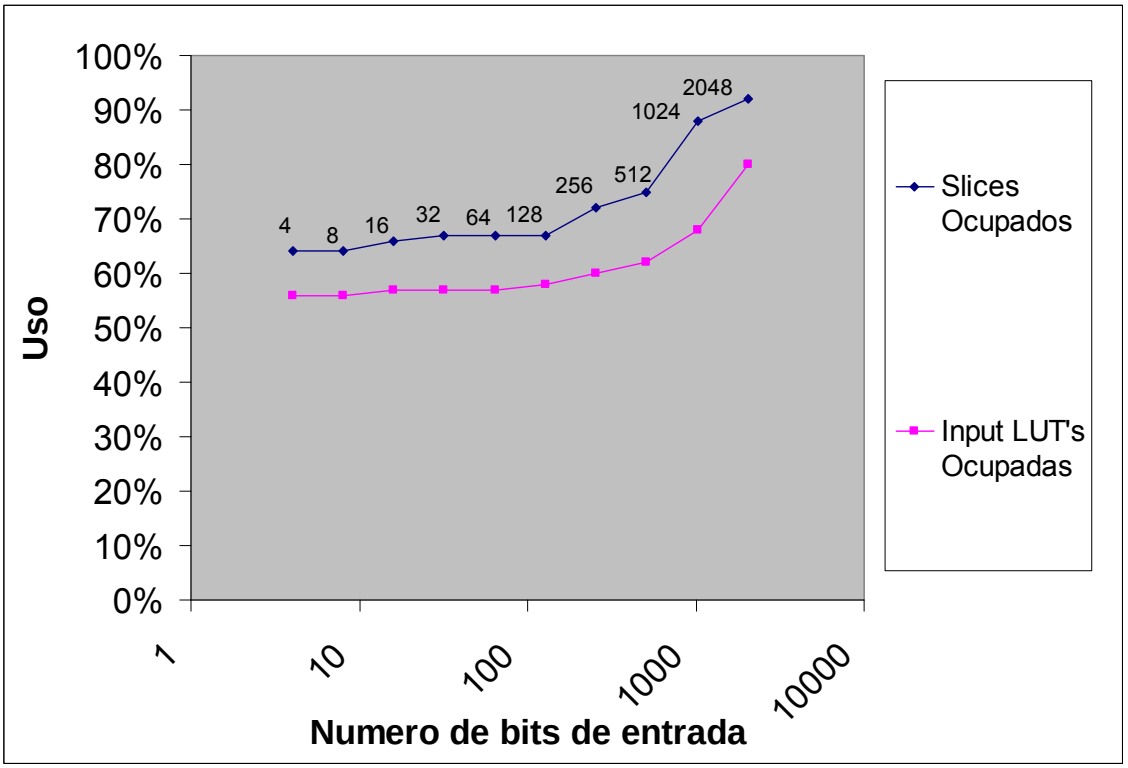
- BHTs de Dirección: 2 BHTs, una denominada “Taken” y otra “Not Taken”. Debido al comportamiento marcado de los saltos, los saltos que tienen a ser Taken se alojaran en la BHT del mismo nombre actuando de modo equivalente la BHT Not Taken; ambas esta indexadas por la misma técnica usada en el predictor GShare, es decir una XOR entre los bits menos significativos de la dirección y el contenido del registro de historia.

El funcionamiento es el siguiente, ante una operación de salto, tanto la BHT Taken como la Not Taken generan su propia predicción, la BHT de Elección es la que se encargará, en función de su propia predicción, de elegir cual de las BHTs de Dirección ofrecerá la predicción definitiva.

A diferencia con las BHTs convencionales, en este caso sólo se actualizará la BHT de Dirección elegida por la BHT de Elección, esta última se actualizará siempre, salvo en los casos en los que de una predicción errónea, y la BHT de Dirección elegida acierte.

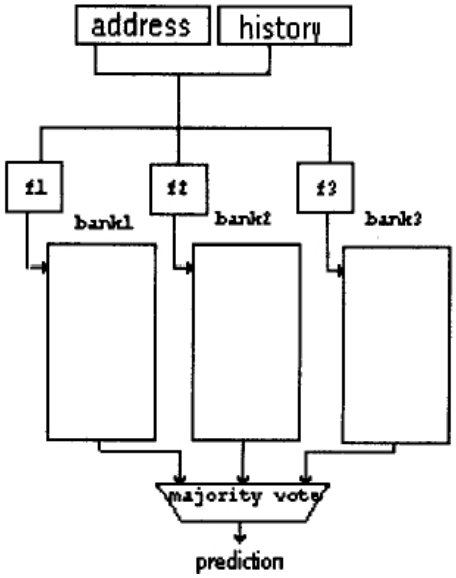
Como resultado de esto, los saltos con tendencia a ser tomados, tendrán su predicción en la BHT Taken, del mismo modo que los saltos con tendencia a no ser tomados tendrán la suya en la BHT Not Taken, esta separación nos asegura que la mayoría del tiempo la información almacenada en la BHT Taken será “tomado”, evitando el efecto destructivo del Aliasing.

Gráfica 6. Porcentaje de uso de la FPGA para el predictor Bi-Mode



Skew

Figura 32. Esquema del predictor Skew



El diseño del predictor Skew [9] se basa en la observación de que la mayor parte del aliasing sucede no por un número reducido de entrada en la PHT sino por la falta de asociatividad en dicha tabla (la mayor parte de los conflictos son por aliasing de asociatividad, no por aliasing de tamaño), lo cual se solucionaría mediante la introducción de una BHT asociativa por conjuntos en el sistema de predicción. Sin embargo, esto introduciría a su vez la introducción de etiquetas, resultando finalmente una solución poco efectiva por su coste.

El predictor Skew trata de emular la asociatividad por conjuntos sin que esto suponga un coste añadido demasiado alto como para descartar su uso. Para ello hace uso de tres predictores de menor tamaño (en nuestro proyecto se encuentran implementados mediante predictores GShare) a cuyas tablas BHT se accede a través de tres funciones de hash distintas. Para obtener el resultado de nuestra predicción utilizaremos un sistema de votación, en el que se predecirá aquella dirección que sea predicha por la mayoría de los predictores.

A la hora de realizar las tres funciones hash, escogemos las propuestas en [9] para la caché asociativa con sesgo, que son las siguientes:

Supongamos H definida como:

H: $\{0, \ldots, 2^{n-1}\} \rightarrow \{0, \ldots, 2^{n-1}\}$
$\{y_n, y_{n-1}, \ldots, y_1\} \rightarrow \{y_n \blacklozenge y_1, y_{n-1}, \ldots, y_1\}$

Donde ♦ es la operación XOR.
Podemos definir ahora las tres funciones hash como las siguientes:

F1: $\{0, \ldots, 2^{n-1}\} \rightarrow \{0, \ldots, 2^{n-1}\}$
$(V_2, V_1) \rightarrow H(V_1) \blacklozenge H^{-1}(V_2) \blacklozenge V_2$

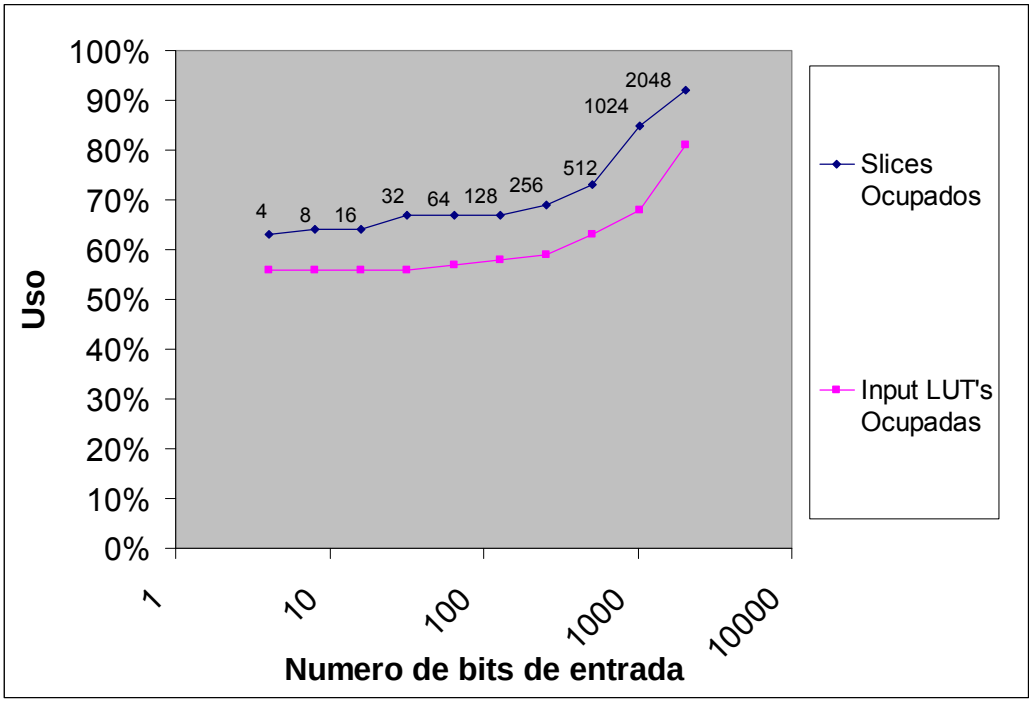
F2: $\{0, \ldots, 2^{n-1}\} \rightarrow \{0, \ldots, 2^{n-1}\}$
$(V_2, V_1) \rightarrow H(V_1) \blacklozenge H^{-1}(V_2) \blacklozenge V_1$

F3: $\{0, \ldots, 2^{n-1}\} \rightarrow \{0, \ldots, 2^{n-1}\}$
$(V_2, V_1) \rightarrow H^{-1}(V_1) \blacklozenge H(V_2) \blacklozenge V_2$

Para aplicarlas a nuestro sistema, tomaremos como V_1 los 2^{n-1} bits menos significativos de la dirección del salto a consultar, y como V_2, 2^{n-1} bits de historia global.

A la hora de actualizar las tablas de los predictores, si la predicción realizada coincide con el sentido tomado por el salto, se actualizarán solo los predictores que hayan acertado. En caso de que la predicción hubiese sido errónea, deberemos actualizar todos los predictores. Con esto logramos una menor interferencia entre saltos que acceden a la misma entrada de la tabla: si la predicción proporcionada por un predictor no era la adecuada, es probable que se deba a que la información en esa entrada estuviese relacionada con otro salto con el que haya aliasing. Así, al no actualizar las entradas que han generado una predicción incorrecta, reducimos la posibilidad de que varios saltos interfieran entre sí.

Gráfica 7. Porcentaje de uso de la FPGA para el predictor Skew



Modulo comparador de estadísticas

El modulo comparador de estadísticas se encarga de preparar los datos necesarios que luego mostraremos por pantalla. Su funcionamiento teórico es simple, cuenta cada salto tomado en función del número de veces que se activan los predictores, esto es, cada vez que en la fase decode se identifica una instrucción como salto condicional.

Tras esto, una vez conocido el resultado del salto, el modulo comparador recibe por su entrada si la predicción es la misma que el resultado y, en caso afirmativo, suma 1 al numero de éxitos del predictor considerado.

La principal característica de este predictor es que funciona siempre un ciclo después de cada predictor. La necesidad de este retraso se basa a que necesitamos considerar los resultados del predictor el ciclo en el que este se actualiza con el resultado del salto. En consecuencia, y ya que el modulo comparador de estadísticas funciona en paralelo a los predictores, era necesario esperar un ciclo para obtener el valor adecuado del predictor. Así pues, todas las señales que necesita, pero sobre todo la señal de 'enable', trabajan con un ciclo de retraso. Se consigue así que los resultados de los predictores funcionen de manera correcta, y siempre trabajando con los datos adecuados.

Controlador de Pantalla

Funcionamiento

Realizar las mediciones de tasas de ejecución y aciertos para cada uno de los predictores considerados en el proyecto no sería una labor completa si no existiese un medio eficaz para comprobar los resultados obtenidos. Con el objetivo de habilitar un sistema como el anteriormente mencionado, se incluye en el diseño del proyecto un controlador de pantalla plenamente funcional, en sustitución del incluido por defecto en la arquitectura del procesador Leon 3.

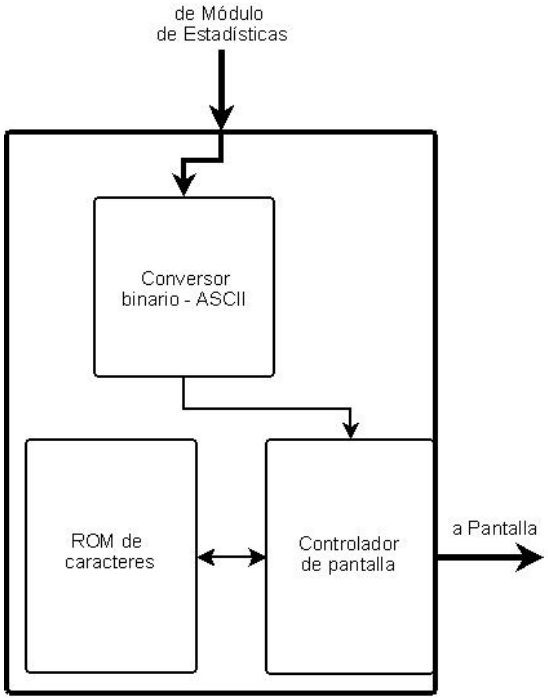
La estructura de este controlador consiste en un controlador central que se conecta a una memoria ROM de caracteres. Éste controlador se conecta a los puertos de salida VGA y

genera las señales oportunas para indicarle a la pantalla qué valores RGB debe mostrar para cada punto. La ROM almacena la codificación para cada carácter, y dado el índice del mismo, devuelve sus valores para ser mostrados.

El diseño del controlador cuenta con una entrada por cada línea que se quiera mostrar por pantalla en la cual se introducen directamente los valores que se quiere representar. La longitud de éstas entradas está parametrizada, de modo que se pueda modificar a conveniencia la longitud de las cadenas de entrada. Si se pretende introducir una cadena más larga que la longitud de línea indicada en el parámetro, dicha cadena será truncada. Si se introduce una cadena más corta, el resto de espacios se rellena automáticamente con espacios en blanco.

Dado que el código de índice de los caracteres en la ROM coincide con su codificación ASCII, y que los datos que el sistema de estadísticas de predictores no tienen el mismo formato, se hace necesario, asimismo, añadir un sistema conversor, que pase los datos numéricos obtenidos directamente en binario (tal y como se obtienen en las estadísticas) a su codificación ASCII para ser mostrados por pantalla. Nuevamente, este conversor consistirá en una memoria ROM que realice la operación de transformación inmediatamente.

Figura 33. Resumen esquemático de las conexiones del modulo de pantalla



Salida por pantalla

Aunque en un principio se planteó la posibilidad de implementar todos los predictores en paralelo, el área de la FPGA está muy limitada, por lo que aunque para predoctores pequeños esta idea era posible, decidimos tratar cada predictor de manera única. Tal y como hemos visto previamente, siempre consideramos datos como área de ocupación y retardos en relojes, por lo que está opción, aun siendo motivada por una limitación, presenta ventajas como por ejemplo una mayor cantidad de datos estadísticos con los que trabajar. De esta manera presentamos dos líneas de datos por pantalla. Cada una comienza con una abreviatura de la información que representa el número a continuación.

La primera, etiquetada como 'TOT>', indica el número total de saltos condicionales considerados en las estadísticas. La segunda línea indica el número de excepciones presentes al ejecutar un programa. Este dato, es muy influyente en los resultados, por lo que nos pareció

destacable su inclusión. La leyenda de este contador es 'EXC>'. Por ultimo, tenemos la línea marcada como 'PRE>', la cual simboliza el numero de aciertos del predictor considerado.

Es necesario advertir que los todos resultados aparecen en codificación hexadecimal y por tanto será necesaria una conversión al sistema decimal antes de poder operar con ellos. Además, en todos nuestros análisis presentamos los datos como porcentaje de aciertos, lo que supone un trabajo posterior con los datos, aunque eso si, este se reduce a considerar el numero de predicciones correctas frente al numero de saltos considerados.

Figura 34. Ejemplo de ejecución de un benchmark

Digital Clock Manager (DCM)

Distintas partes del diseño del procesador (entre las que se incluye el controlador de pantalla) requieren la utilización de señales de reloj funcionando a diferentes frecuencias. Una manera de enfrentarse a este problema es utilizar lógica secuencial conectando la señal global de reloj a la entrada de multiplicadores y divisores de frecuencia diseñados con biestables y lógica combinacional. Sin embargo, esta aproximación implica la aparición de fenómenos anómalos y difíciles de controlar como skew (retardo medio entre dos señales sincronizadas) y jitter (retardo entre la transición esperada y la real). En resumen, ocasionan problemas de ruido.

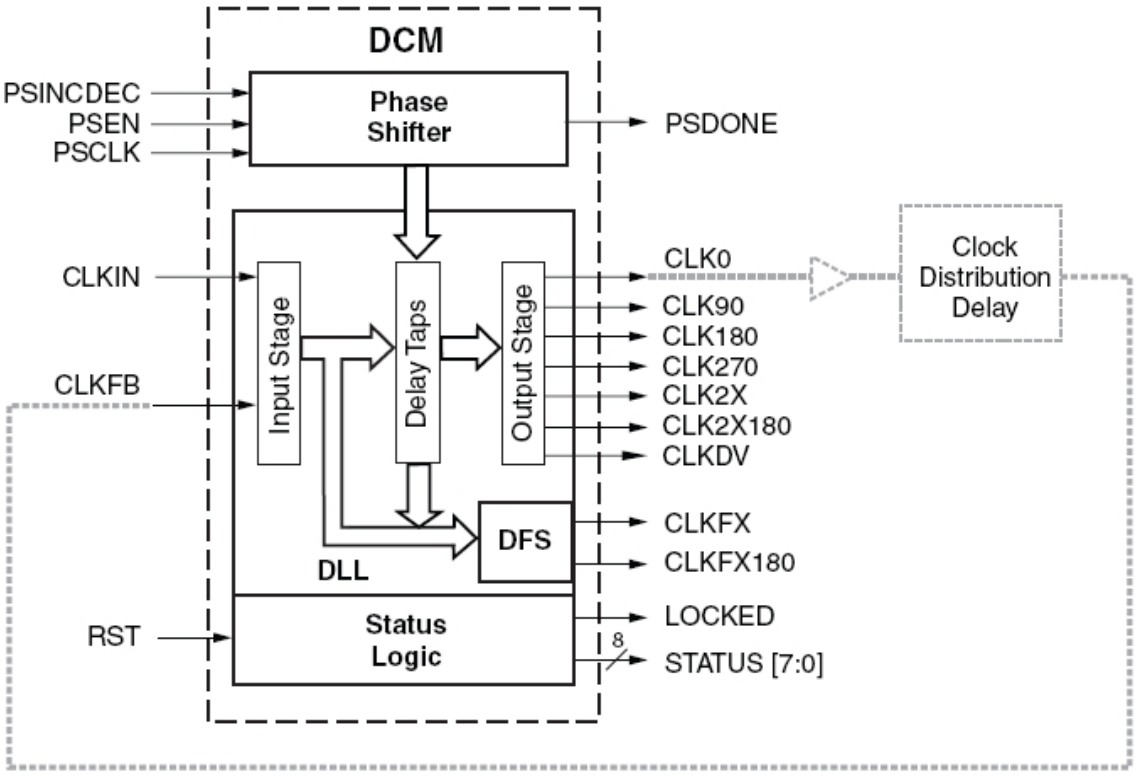
Para obtener una respuesta con mejor rendimiento y eliminar, en la medida de lo posible, este tipo de riesgos, la placa de prototipado Virtex II PRO XUP incluye una serie de módulos denominados Digital Clock Managers (o DCM) destinados a proporcionar una señal fiable de reloj, eliminando todas las anomalías que se pudiesen generar.

Un sistema de este tipo permite interactuar de múltiples maneras con la señal de reloj del sistema. Así, por ejemplo, podremos desfasar el pulso en 0º, 90º, 180º y 270º, eliminar retardos en la señal de reloj, o modificar su frecuencia, multiplicándola o dividiéndola. En nuestro caso, será ésta última función la que más nos interesará.

El procesador SPARC utiliza, para su funcionamiento interno, un DCM a través del cual modifica la frecuencia de la señal global que gobierna el procesador. Los parámetros de multiplicación y división de la frecuencia son ajustables desde el menú de configuración del procesador xconfig, y para la configuración por defecto se encuentran fijados a unos valores de multiplicación por 13 y división por 20. Esto, aplicado a la señal de reloj global de la placa Virtex II Pro XUP (que funciona a 100MHz) nos proporciona una frecuencia de 65 MHz con la que trabajar. Dado que nuestro monitor necesita una señal de reloj de menor frecuencia (en las

tablas de referencia del manual de Virtex II PRO XUP [1] comprobamos que para una salida por pantalla de 640x480 pixels como la deseada, necesitamos un reloj funcionando a 25MHz) tomaremos la salida del DCM que mantiene la misma frecuencia que a la entrada y la dividiremos, ya en el interior del módulo de pantalla, entre cuatro, con el objetivo de obtener la frecuencia deseada para el manejo del controlador.

Figura 35. Estructura de un DCM



Archivos modificados del Leon 3

A la hora de utilizar el diseño en VHDL disponible para el Leon 3, nos vimos obligados a hacer una serie de modificaciones para poder integrar nuestro proyecto. A continuación exponemos esos cambios, así como su motivación y utilidad. Es importante desacatar que más allá de la utilidad xConfig explicada anteriormente, no se ha realizado ningún cambio al código original del Leon 3, salvo las adicciones necesarias para integrar nuestros módulos. De la misma manera, ha sido necesario añadir la señal de ResetCont al diseño UCF original, para poder usar el botón elegido en la FPGA a este propósito (boton arriba dentro de los botones de movimiento).

Modulo principal – *'Leon3mp.vhd'*

En el modulo principal añadimos el port map al modulo de pantalla. A este le pasamos la señal 'Estadísticas', dentro de la cual mandamos tanto los saltos considerados como los aciertos del predictor estudiado en cada caso. Además le añadimos la señal de Reset ya comentada, así como el reloj y las señales que servirán como salida por pantalla. Estas señales son heredadas del diseño original del Leon 3, por lo que no es necesario ni crearlas ni definirlas en el UCF.

Modulo de pipeline de la unidad de enteros – *'iu3.vhd'*

Este modulo contiene la mayoria de los cambios hechos al Leon 3, y en realidad son señales que se comparan para identificar las instrucciones de salto condicionales (BICC), ignorando aquellas de salto incondicional, que como ya hemos explicado, para los predictores de salto no se consideran.

Aquí hemos incluido además el modulo PredictoresSalto.vhd, con los predictores o el predictor en función del tamaño elegido (como vimos al comienzo de la memoria, las limitaciones en este aspecto han influido poderosamente en el proyecto).

Librería de configuración – *'Leon3.vhd'*

Esta librería nos ha servido para añadir el tipo Estadísticas, de manera que fácilmente podemos añadir nuevos datos y reducir el tamaño de los mismos, sin tener que cambiar ninguna entidad del Leon 3. Así mismo, hemos aprovechado el tipo existente IRQ_OUT_TYPE, para añadir la señal de estadisticas. El motivo principal una vez más es la sencillez, puesto que conseguimos que las unidades intermedias no se vean afectadas.

Librerias intermedias para la declaración de tipos y entidades – *'Leon3cg.vhd', 'Leon3s.vhd', 'libiu.vhd', 'libproc3.vhd', 'proc3.vhd'*

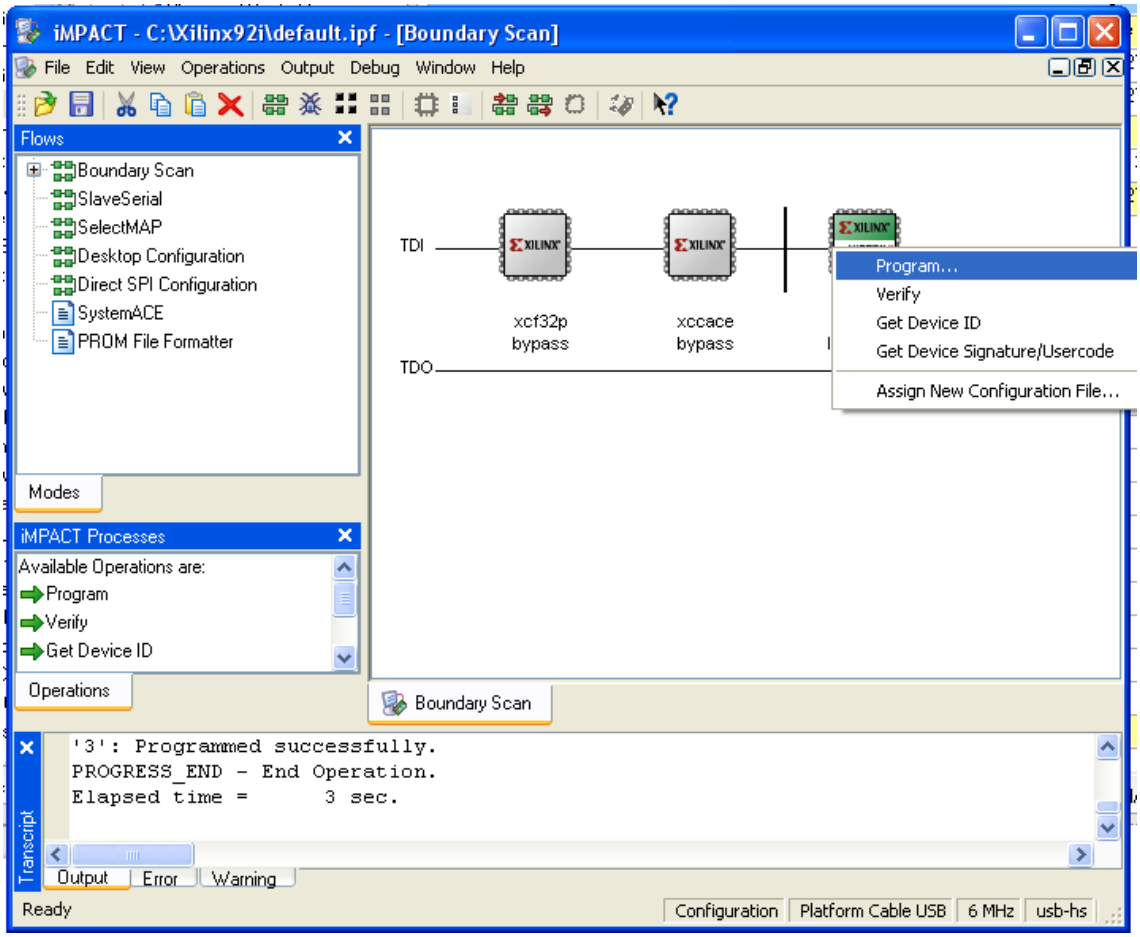
Hemos añadido la señal de ResetCont, que mandamos desde el modulo principal hasta el nivel mas bajo del pipeline de la unidad de enteros. Por simplicidad todos nuestros diseños trabajan con una señal Reset que se activa cuando vale '1', sin embargo, tanto la placa como el diseño original del Leon 3 funcionan con Reset a '0'.

EJEMPLO DE EJECUCION

Para poder cargar, ejecutar y medir los porcentajes de éxitos de cada predictor, fue necesario sintetizar varios proyectos diferentes dentro del Xilinx ISE. Cada archivo bitstream (con extensión .bit) se incluye en el CD adjunto, con un nombre descriptivo del tamaño y del predictor considerados. Una vez generado cada .bit, se carga mediante la herramienta iMPACT de Xilinx.

Para ello destacamos que mediante este software se reconoce mediante el cable USB adecuado la placa FPGA. De esta manera, al detectar la FPGA VirtexII Pro, pedirá tres archivos. Los dos primeros son opciones de control que no hay que definir, por lo que presionaremos 'bypass'. La tercera es la carga del .bit adecuado. Una vez elegido, presionamos en el boton derecho dentro del ultimo elemento detectado y seleccionamos 'Program…"

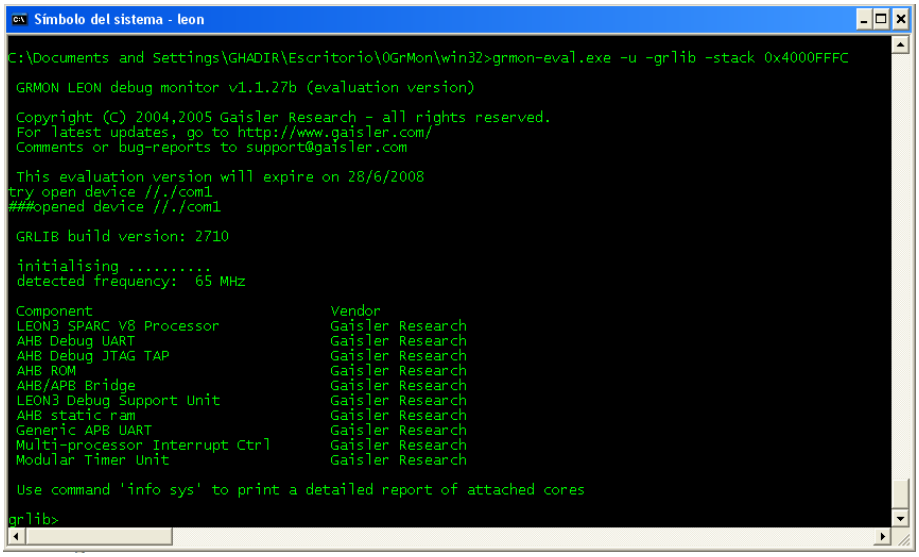
Figura 36. Programa de carga de .bit iMPACT



Tras esto, ejecutamos la consola de comandos de MS-Dos y nos posicionamos en la ruta adecuada, dentro de la cual esta el programa GR-Mon (en nuestro caso al tratarse de una versión de evaluación se llama 'grmon-eval.exe'. Una vez dentro escribimos la sentencia:

'grmon-eval.exe –u –grlib –stack 0x4000FFFC'

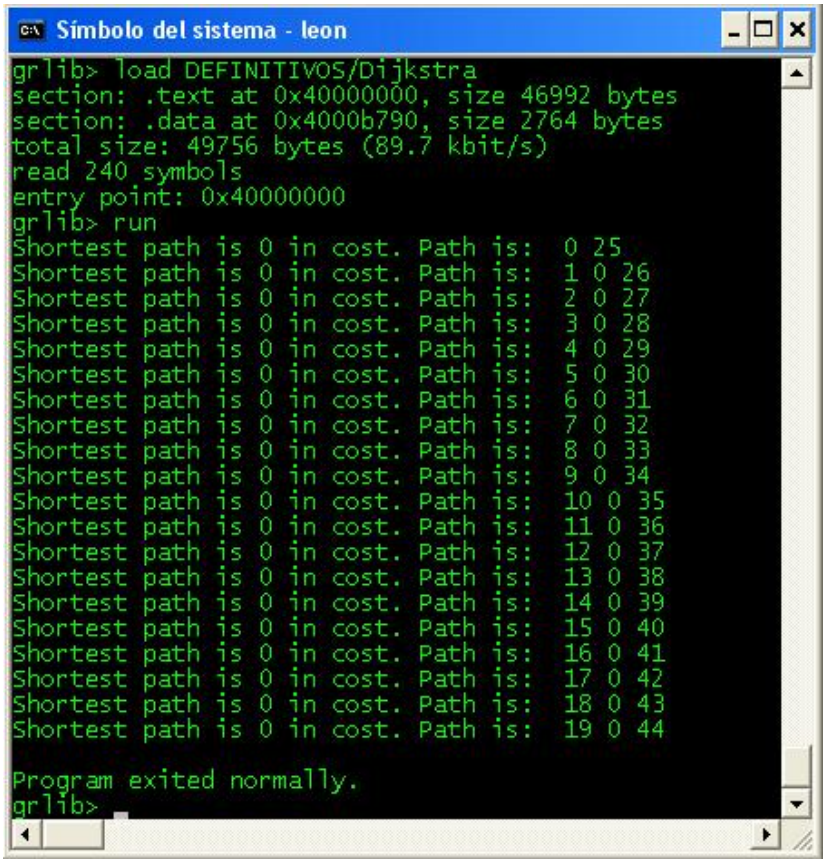
Figura 37. Conexión correcta con el Leon 3 cargado en la FPGA



Siendo el significado de las opciones: activar el modo UART-Loopback (-u), con esto conseguimos que la salida por pantalla nativa se muestre a través de la consola en el ordenador que esta ejecutando el grmon. Preparar para Leon 3 (-grlib), se activa esta opción por seguridad. Posicionar la pila de programa en una dirección correcta (–stack 0x4000FFFC).

Por ultimo, ejecutamos la sentencia load X, donde X es el archivo o ruta al archivo compilado para la arquitectura SPARC v8 adecuado (ver apéndice "*Herramientas y Método de compilación de aplicaciones para el Leon 3*"). Una vez cargada, escribimos 'run' y trabajamos con los datos resultantes.

Figura 38. Ejemplo de ejecución de una aplicación en el Leon 3



RESULTADOS EXPERIMENTALES

Breve explicación de los resultados

Por su utilidad, todos los resultados experimentales están representados como gráficas, donde podemos observar el porcentaje de aciertos para cada benchmark y para cada predictor según el número de entradas. Tras la correcta medición de estos datos y una vez acabada esta breve introducción explicaremos como para según que tipo de operaciones se observa una mejor disposición de cierto predictor, de manera que sirva al propósito general del proyecto, dando lugar a una herramienta útil de medición del comportamiento y una proyección general del mismo, lo que servirá a la elección correcta dentro de una arquitectura especifica.

Se puede observar que todos los predictores, excepto el Filter y el Local, han sido evaluados hasta tamaños de 2048 entradas, mientras que los otros dos solo llegan hasta 256 y 1024 entradas, respectivamente. Esto se debe a las restricciones de espacio de la FPGA con la que trabajamos. Se trata de predictores con tablas muy grandes, que llegados a un punto determinado, no caben en la placa de prototipado.

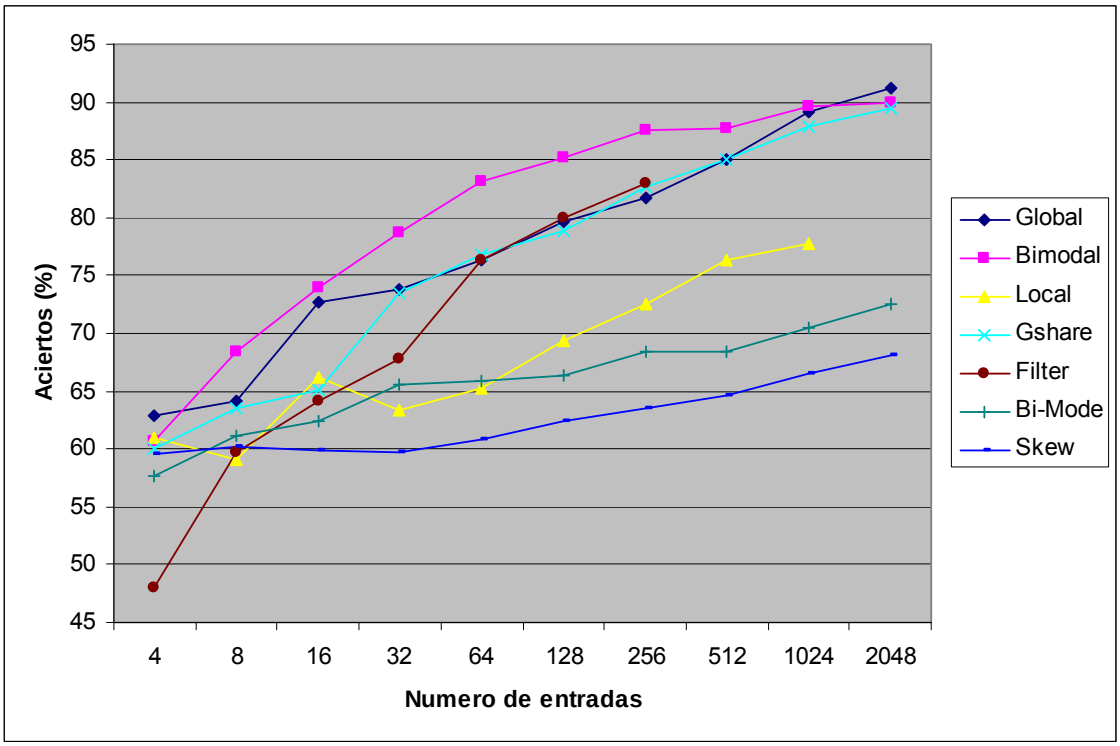
Estos resultados se han tratado de manera casi-exacta, debido a una divergencia de resultados observada gracias a varias medidas de control de validez en los mismos, implementadas por nosotros. Esta divergencia esta perfectamente identificada, y es consecuencia de excepciones semi aleatorias dentro de las instrucciones previas de carga de librerías propias de la arquitectura SPARC v8. Sin embargo, la magnitud de esta diferencia ronda en el peor caso el 1 por 1000 (instrucciones), frente al mejor caso, que se corresponde al 1 por 250.000. Esta divergencia está controlada en el porcentaje de aciertos, y aunque presenta esta diferencia, los datos son suficientemente precisos como para aceptar la validez de los mismos.

Por ultimo, antes de empezar con el análisis caso a caso de los benchmarks, destacamos dos datos a tener en cuenta. En primer lugar, para comprobar que los predictores funcionaban correctamente se han realizado tanto bancos de prueba en ModelSim, como bancos de pruebas básicos en C, es decir fragmentos de código preparados para forzar errores y aciertos en los predictores que cargamos directamente en el Leon 3, de manera que nos permitía comprobar el buen funcionamiento. Segundo, observamos que para ciertos benchmarks, tales como por ejemplo el Search o el Sha, el porcentaje de éxitos es muy alto en todos los casos. Estos resultados también son útiles dado que permiten al diseñador seleccionar un predictor sencillo con un coste HW muy bajo.

Con estas consideraciones previas observamos los siguientes resultados:

Estadísticas para BasicMath

Gráfica 8. Porcentaje de aciertos para BasicMath

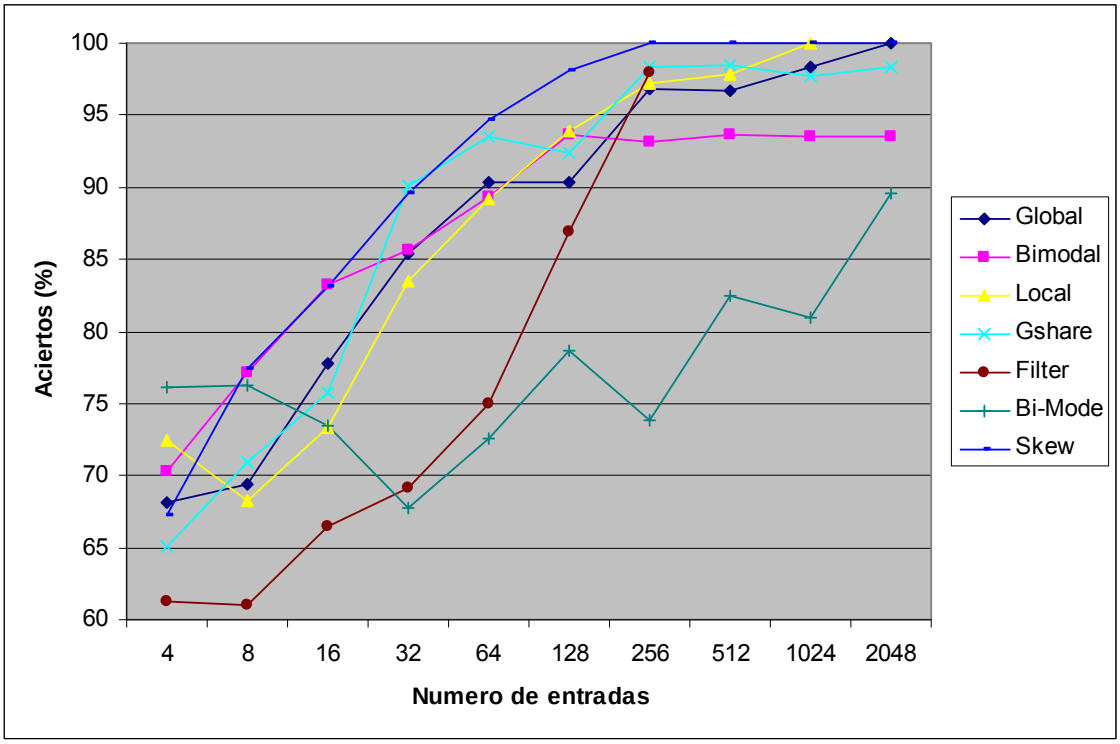


En el caso de este predictor podemos observar como para tamaños similares son más rentables tanto en coste económico como en espacio en placa predictores como el Bimodal o el Global. Sin embargo, vemos como hay una tendencia de crecimiento en el caso del Gshare o el Local que hace pensar que para entornos más grandes y con mayor carga de trabajo sería una mejor opción, puesto que por ejemplo el comportamiento del Bimodal se estanca y no mejoraría para tamaños más grandes.

Es igualmente destacable el caso del Filter, cuyo crecimiento en el porcentaje de aciertos al aumentar el número de entradas es muy pronunciado, lo que hace pensar que aunque es más caro que los demás, para entornos de aplicaciones con una grandísima carga de trabajo en aritmética en punto flotante funcionaria muy adecuadamente.

Estadísticas para DhryStone

Gráfica 9. Porcentaje de aciertos para DhryStone

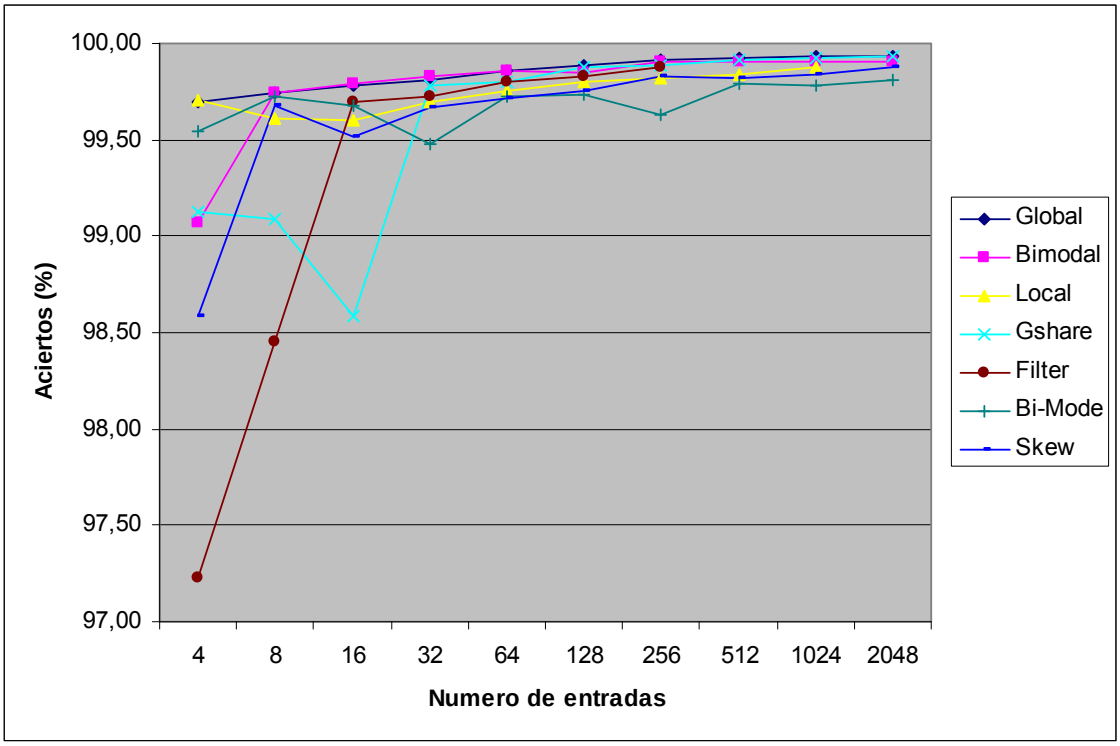


Para el DhryStone podemos comprobar que la elección del predictor puede tener una influencia muy importante en el rendimiento del sistema, dado que predictores de similar coste tienen una eficacia muy distinta (con diferencias de hasta un 30% en la tasa de aciertos). Además destacamos el caso del Bimodal, que ya en este caso no representa la mejor elección, debido sobre todo a la cantidad de instrucciones aritméticas presentes en este benchmark. Por otra parte, el sistema cumple su propósito, al mostrar por ejemplo como predictores Bimodales de más de 128 entradas no mejoran el rendimiento o como la ganancia del Bi-Mode o el Filter aumenta considerablemente con su tamaño.

A la vista de los resultados, podemos concluir que la mejor elección para este benchmark sería emplear un Skew de 256 entradas, o en su defecto un Local de 1024, siendo preferible el primero debido a su menor coste en espacio. En caso de estar dispuestos a sacrificar rendimiento por espacio, un Bimodal de 128 podría ser una solución a considerar.

Estadísticas para Dijkstra

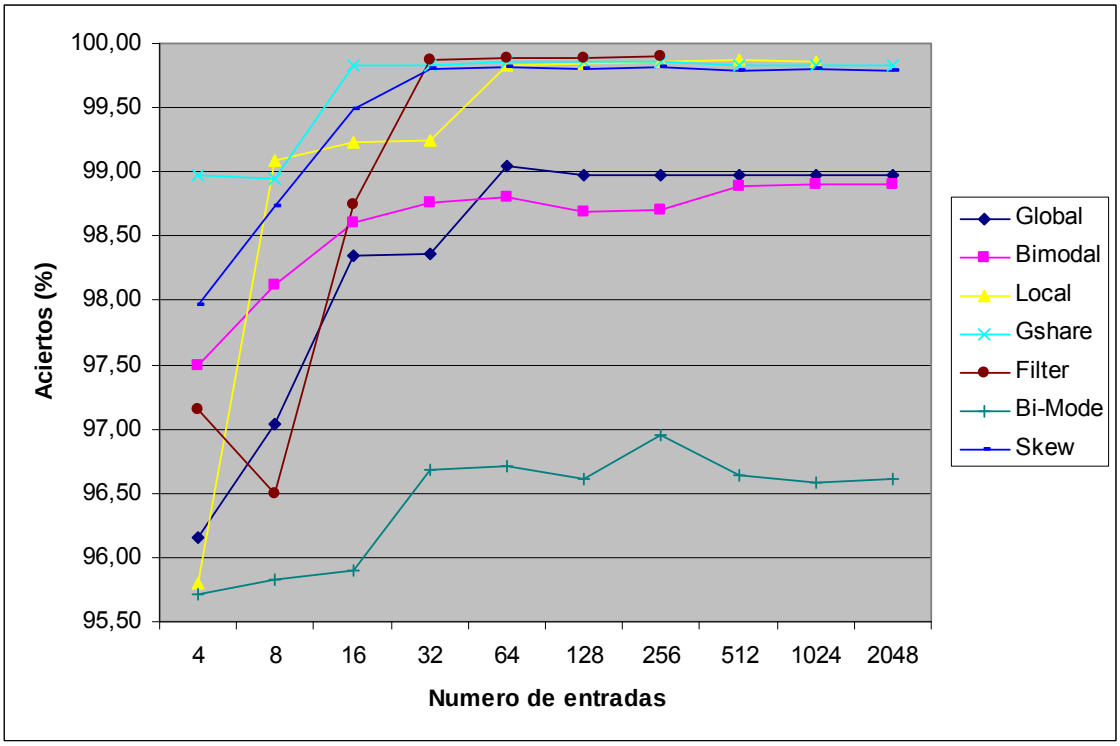
Gráfica 10. Porcentaje de aciertos para Dijkstra



Este benchmark presenta un comportamiento muy diferenciado para pequeños tamaños, sin embargo, a medida que estos crecen, las diferencias se reducen, lo que provoca que en este tipo de aplicaciones la elección de un predictor no sea muy compleja, orientándose sobre todo a la reducción de costes.

Estadísticas para QuickSort

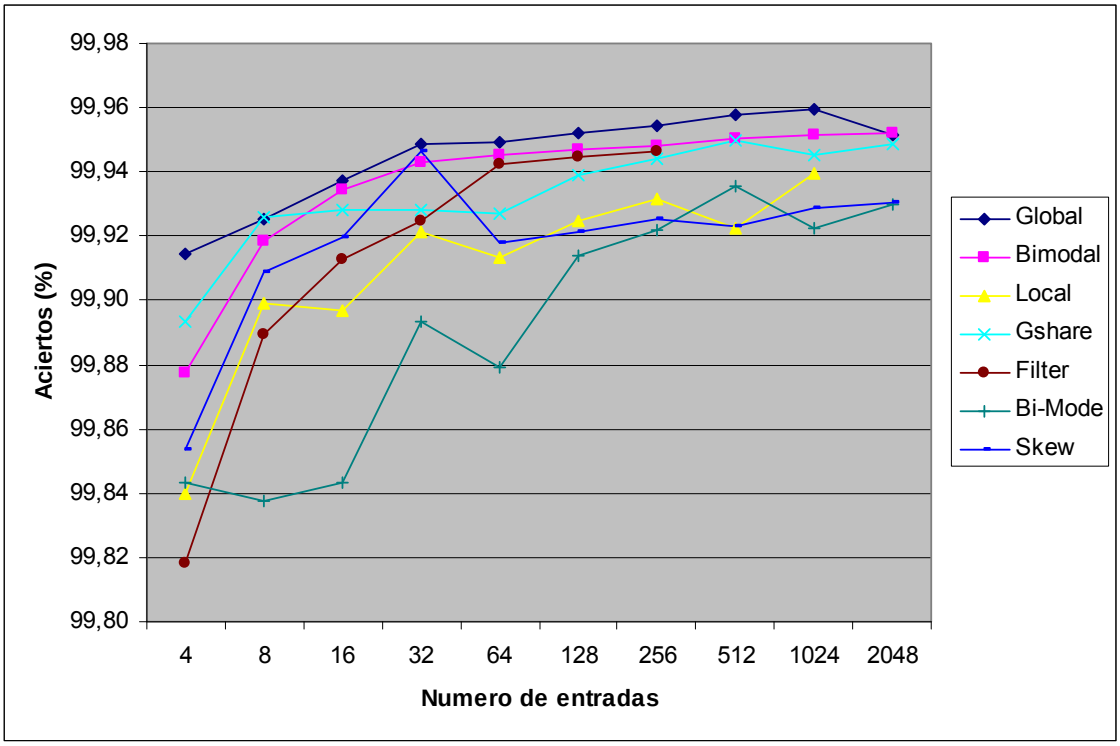
Gráfica 11. Porcentaje de aciertos para QuickSort



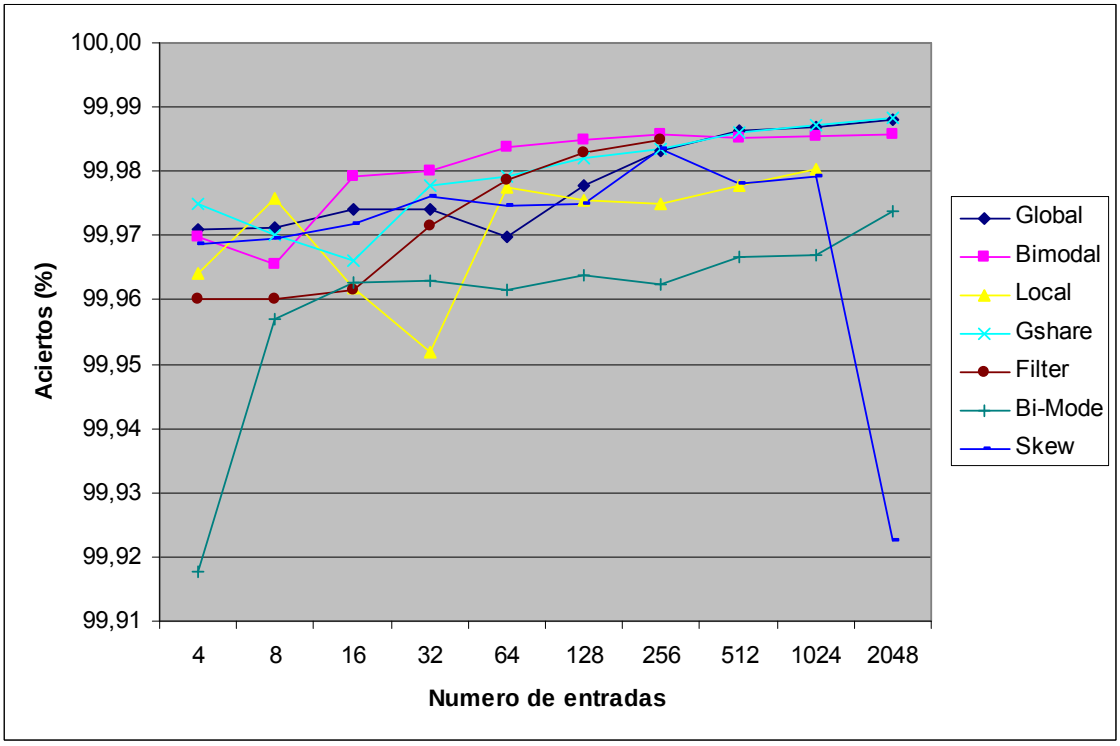
Para el algoritmo QuickSort podemos comprobar cómo predictores relativamente simples, como GShare o Local, proporcionan unos resultados casi óptimos, que otros predictores más complejos igualan, como Filter, o no son capaces de alcanzar, como es el caso de Bi-Mode. Ante una situación así, es evidente que la elección de uno de los predictores simples es la más adecuada. El ahorro en espacio y coste aumenta más todavía al darnos cuenta de que el número de entradas que necesitarán los predictores sencillos para obtener esos resultados casi idóneos es menor que el que requerirían los predictores complejos.

Estadísticas para BitCount, SearchString y Sha

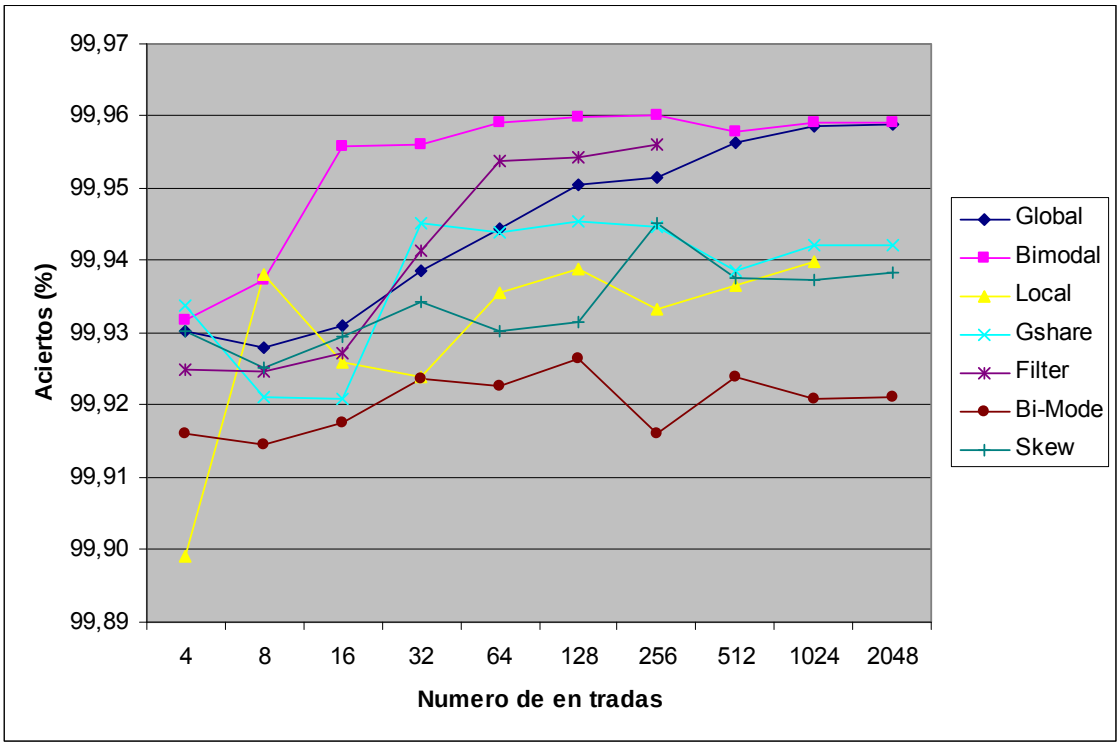
Gráfica 12. Porcentaje de aciertos para BitCount



Gráfica 13. Porcentaje de aciertos para SearchString



Gráfica 14. Porcentaje de aciertos para Sha



En el caso de estos tres benchmarks podemos observar cómo cualquiera de los predictores proporcionan muy buenos resultados desde tamaños muy pequeños, por lo que en estos casos, la introducción de un predictor complejo o de alto número de entradas no estaría en absoluto justificada. En esta situación, el uso de un predictor básico y de tamaño reducido sería más que suficiente en caso de que el perfil de programa que fuese a ejecutarse en nuestro procesador fuese de este tipo.

CONCLUSIONES

A la luz de los resultados experimentales que hemos expuesto previamente, podemos afirmar que la plataforma que hemos diseñado en este proyecto es un sistema eficaz para la evaluación de cualquier predictor de saltos una vez implementado a partir de un código en VHDL. Habiendo conseguido integrarlo de manera eficiente sobre la arquitectura SPARC v8, nuestro sistema funciona utilizando el reloj del propio procesador. En consecuencia, nos encontramos con que en la implementación actual podemos trabajar a frecuencias que oscilan entre los 65 MHz y los 100 MHz para la FPGA tratada, llegando a alcanzar los 400 MHz en otros soportes HW.

Con ésta plataforma de evaluación obtenemos una reducción de hasta tres órdenes de magnitud en el tiempo requerido para evaluar correctamente un predictor de saltos. Al estar basada en una arquitectura SPARC v8 segmentada capaz de ejecutar una instrucción por ciclo, nuestra plataforma de evaluación trata como mínimo 65 MInst/seg, mientras que el sistema de simulación por software SimpleScalar, en la máquina de referencia (Pentium 4 a 1,6 GHz), ejecuta del orden de 200 KInst/seg. Además, el coste en área de la implementación con el predictor más pequeño conectado es de menos del 2% de la FPGA, por lo que en éste aspecto también la plataforma resulta muy rentable.

Del mismo modo, las pruebas que llevamos a cabo con los distintos predictores y benchmarks han mostrado que, en determinadas ocasiones, la elección de un predictor incorrecto puede conllevar una tasa de fallos muy elevada. Igualmente, podemos detectar

rápidamente el punto en que la tasa de aciertos deja de crecer al seguir aumentando el número de entradas de las tablas de los predictores, marcando así el punto a partir del que deja de resultar rentable ampliar el tamaño de los predictores. Gracias a esto podemos determinar, para cada aplicación, qué predictor es el mejor y cual es su tamaño óptimo, maximizando la relación entre rendimiento y coste HW.

Por último, el hecho de que al trabajar con herramientas de diseño disponibles de manera gratuita y código abierto de licencia GPL, cumplimos ampliamente nuestro objetivo inicial de proporcionar un marco de trabajo ampliable a muy bajo coste (únicamente el asociado a la adquisición de la FPGA) y fácilmente accesible por cualquier investigador o desarrollador interesado en continuar el trabajo que aquí hemos presentado.

TRABAJO FUTURO

Uno de los principales objetivos que nos planteamos a la hora de elegir un proyecto de Hardware es lograr diseñar un modelo cuya proyección no se quedase únicamente en el propio proyecto. Si la informática es una ciencia que avanza más rápidamente que ningún otro campo técnico, dentro de esta es indudable que el hardware es el que mas sufre el paso del tiempo. Desde la famosa regla de Ley de Moore, observamos como el avance en este campo es poco menos que imparable.

Por tanto, resultaba importante enfrentarnos a un reto que no finalizase con nosotros, sino que pudiese continuarse de cara a ofrecer mejoras básicas, si bien no en sistemas de alto rendimiento, si en sistemas empotrados o para conocer mejor el funcionamiento de los predictores de salto frente a determinados tipos de aplicaciones.

Una de las principales y más rápidas mejoras a nuestro diseño es obvia, añadir más tipos de predictores de salto al diseño, de manera que pueda comprobarse su efectividad frente a los siete implementados por nosotros. La falta de tiempo y la complejidad en algunos puntos al trabajar con el Leon 3 nos privó de este trabajo, no obstante, siempre hemos tratado de diseñar nuestro código para facilitar este trabajo futuro a corto plazo, llegando incluso a reducir todo a un simple port map en vhdl del modulo que quiera utilizarse.

Por otro lado, otra mejora ciertamente posible es no limitar el uso de los predictores al cálculo de estadísticas, sino implementarlo directamente en el SPARC v8 para mejorar su rendimiento (como explicamos anteriormente, esta es fundamentalmente la razón del uso de los predictores de salto). Así mismo, se plantea la posibilidad de instalar el sistema operativo Linux disponible para ésta arquitectura y de este modo generalizar la evaluación de predictores no solo a benchmarks concretos sino a cualquier aplicación disponible para éste entorno.

Sin embargo, todo esto son mejoras a medio-largo plazo. Pero la principal motivación de nuestro proyecto, como expusimos previamente, se basa en servir de herramienta de medición y comparación para otras arquitecturas. Pensando por ejemplo que tipo de predictor va a ser mas eficiente en arquitecturas empotradas que manejen por ejemplo gran cantidad de procesamiento de datos enteros, o algo más preparado para entornos de oficina, de eventos comerciales (y en general de cara al publico) o por ejemplo para sistemas de posicionamiento GPS que traten más datos en punto flotante.

Nuestro proyecto pretende ser una herramienta de salida para más vías de trabajo, desde grandes proyectos hardware que alteran la ruta de datos del procesador sobre el que se implementa a la prueba y elección de un predictor adecuado hasta pequeñas implementaciones en sistemas empotrados que se enfocan a un objetivo muy diferenciado. De este modo, cualquier ampliación del proyecto podría realizarse siempre restringiéndonos a las limitaciones de la FPGA elegida para las pruebas y la capacidad del software utilizado, pero usando las herramientas y ventajas que el software posee, que permitirían, por ejemplo, adecuar el diseño a una FPGA de mayor capacidad sin excesivo esfuerzo gracias al uso del configuración del Leon 3 preparado a este efecto.

Asimismo, esta misma aproximación para la evaluación de predictores de salto puede aplicarse a otros aspectos de la arquitectura de un procesador, permitiendo ampliar el espacio de diseño de este. Basándose en una plataforma de evaluación similar a la desarrollada aquí, pueden desarrollarse evaluadores para, por ejemplo, adaptar el tamaño de las colas de load/stores que gestionan las comunicaciones con memoria, o distintas estrategias de renombramiento de registros, elementos de alto coste y complejidad cuya optimización permite importantes ahorros en espacio y consumo.

Por ultimo, gracias al reciente auge de las arquitecturas multiprocesador, otra modificación posible de medio a largo plazo sería la utilización de nuestros predictores para que trabajasen en paralelo junto a otros procesadores. Para ello, una primera idea sería la creación de un modulo superior que tratase cada procesador de manera independiente, y al acabar sumase los éxitos de cada modulo predictor dentro de cada procesador. Así mismo, se plantea la posibilidad de incorporarlo en HW donde se usen predictores adaptativos, de manera que sea posible, mediante una etiqueta, el uso de diferentes predictores de salto según la aplicación que se vaya a ejecutar, decidiendo esto mediante hardware.

GLOSARIO DE TÉRMINOS

Asíncrono:
Hace referencia al suceso que no tiene lugar en total correspondencia temporal con otro suceso. Si se refiere a una comunicación asíncrona, ésta es en la que se realiza el envío de datos sin la sincronización de un reloj externo.

Bit:
Dígito en el sistema binario.

Bus:
Conjunto de conductores eléctricos en forma de pistas metálicas impresas sobre la placa base del computador, por donde circulan las señales que corresponden a los datos binarios del lenguaje máquina. Si el bus es de datos, este conectará dos dispositivos hardware.

Branch Delay Slot:
Ver "Hueco de salto".

Byte:
8 bits.

Chip:
Circuito integrado o pastilla en la que se encuentran todos o casi todos los componentes necesarios para que un ordenador pueda realizar alguna función.

Ciclo:
Un ciclo es la distancia temporal entre el principio y el final de una onda completa.

Circuito impreso:
Medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de rutas o pistas de material conductor, grabados desde hojas de cobre laminadas sobre un sustrato no conductor.

Decode:
Segunda etapa del pipeline de la unidad de enteros del Leon 3, en esta etapa se decodifica la instrucción contenida en el PC usando los datos que recibe de la cache de instrucciones.

Depuración:
Proceso de mejora de un sistema hasta que realiza su función correctamente.

Empotrado:
Ubicado dentro del chip.

Esquemático:
Es un diagrama, dibujo o boceto que detalla los elementos de un sistema.

Evento:
Acontecimiento ocurrido en el sistema.

Fetch:
Primera etapa del pipeline de la unidad de enteros del Leon 3, en esta etapa se el del PC la siguiente instrucción a ejecutar.

FIFO:
First In, First Out. Protocolo que implementa una cola: lo que viene primero, se maneja primero, lo que viene segundo espera hasta que lo primero haya sido manejado, etc.

Flanco:
Transición del nivel bajo al alto (flanco de subida) o del nivel alto al bajo (flanco de bajada) de una señal.

FPGA:
Dispositivo donde se pueden programar infinidad de diseños digitales distintos.

Grafo:
Conjunto de nodos relacionados mediante aristas de simple o doble sentido.

Hardware:
Es un término general usado para describir artefactos físicos de una tecnología.

Hueco de salto:
Técnica de mejora hardware que se basa en la necesidad de tener siempre algún dato en cada fase de un procesador segmentado. El compilador elige la instrucción que se ejecuta inmediatamente después de una de salto, de manera que mientras el salto se evalúa, la instrucción siguiente se ejecuta siempre. La elección, por tanto de esta instrucción no es algo trivial, y cuando no se puede elegir ninguna que no afecte al resultado final del programa, el compilador la rellena con 'NOP's.

PC:
Registro contador de programa (Program Counter), es el registro más básico dentro de un computador, en el se almacena la dirección de la siguiente instrucción a ejecutar por el procesador.

Pipeline:
Ruta de datos segmentada en la que se divide un procesador para mejorar la eficiencia, en el caso del Leon 3 consta de 7 etapas: Fetch, Decode, Register Access, Execution, Memory, Exception y WriteBack.

Planificación:
Modelo de ejecución para una tarea.

Predecesor:
Que se encuentra con anterioridad en la jerarquía.

Prototipo:
Ejemplar original o primer molde en que se fabrica un diseño.

NOP:
Siglas que se refieren a un tipo común de instrucción en la que no se hace nada. Su uso se basa para servir al procesador de comodín, mientras trata otro tipo de instrucciones sin verse afectado el resultado de cada programa.

Síncrono:
Hace referencia al suceso que tiene lugar en correspondencia temporal con otro suceso. Si se refiere a una comunicación síncrona, ésta es en la que se realiza el envío de datos bajo la sincronización de un reloj externo.

Speed-up:
Cociente que nos dice la mejora de rendimiento obtenida.

Subtarea:
Cada una de las partes de una tarea con significado propio.

Sucesor:
Que se encuentra con posterioridad en la jerarquía.

Unidad reconfigurable:
Parte de la FPGA que puede ser reconfigurada de modo independiente.

VHDL:
Very High Speed Integrate Circuit Hardware Description Language, es un lenguaje de descripción y modelado, diseñado para describir la funcionalidad y la organización de sistemas *hardware* digitales, placas de circuitos, y componentes.

BIBLIOGRAFÍA

[1] Xilinx University Program Virtex-II Pro Development System. Hardware Referente Manual. UG069 (v1.0) March 8, 2005.

[2] J. Hennessy, D. Patterson, "Computer Architecture: A Quantitative Approach" (4th edition), Morgan Kaufmann Publishers, Inc. 2007.

[3] A.N. Eden, T. Mudge "The YAGS Branch Prediction Scheme", Proc. 31st Ann. Int. Symp. on Microarchitecture, 1998.

[4] S. McFarling. "Combining Branch Predictors". Technical Report TN-36, Digital Western Research Laboratory, Junio 1993.

[5] P. Michaud, A. Seznec, R. Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors," in Proc. of the 24th Ann. Int. Symp. on Computer Architecture, pp. 292--303, 1997.

[6] P. Michaud, A. Seznec, R. Uhlig, "Skewed branch predictors", Tech. report, IRISA publ. int. 1031, Junio 1996.

[7] P.Y. Chang, M. Ever, Y.N. Patt, "Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference", Proc. 1996 Conf. On Parallel Architectures and Compilation Techiques, Octubre 1996.

[8] A. Arfel "Plataforma de Comparación para la Predicción Dinámica de Saltos en Hardware FPGA".

[9] P. Michaud, A. Seznec, R. Uhlig, "Skewed Branch Predictors", Rapport de Recherche nº 2987, INRIA, Septiembre 1996.

[10] C.C Lee, I-Cheng K, Chen, and Trevor N .Mudge "The Bi-Mode Branch Predictor", Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, 1997.

[11] A. Stud "Branch Prediction".

[12] A. Ramirez, "Branch Prediction Techniques", UPC-Barcelona, Julio 08.

[13] J.I Magnatti, J.P Sandoval, N.M Lerendegui", Eficacia de los Predictores de Salto en Procesadores", Instituto Tecnológico de Buenos Aires, Departamento de Electrónica.

[14] Gaisler Research, "GRLIB IP Core User`s Manual, Version 1.0.17", Noviembre 2007.

[15] Gaisler Research, "GRLIB IP Library User`s Manual, Version 1.0.17".

[16] SPARC International Inc, "The SPARC Architecture Manual Version 8", 1992.

[17] Xilinx Inc., Virtex-II Pro and Virtex-II Pro X Platform FPGAs:Complete Data Sheet, Xilinx, San Jose, Calif, USA, 2005.

[18] M. R. Guthaus, J. S. Ringenberg, D. Ernst, "MiBench: A free, commercially representative embedded benchmark suite", WWC-4. 2001 IEEE Int. Work. on Workload Characterization, 2001.

[19] Gaisler Research, "The LEON Processor User's Manual, Version 2.3.7", Agosto 2001.

[20] T. Y. Yeh and Y. N. Patt. “Alternative implementations of two-level adaptive branch prediction”. In Proc. 19th Int. Sym. on Computer Architecture, Mayo 1992.

[21] R. Williams, “SimpleScalar CPU Simulator”, BSc CRTS/CSE yr4, 10/05, 10/06

[22] P. Bhojwani “SimpleScalar Introduction for toolset release v2.0”

APÉNDICES

HERRAMIENTAS Y METODO DE COMPILACION DE APLICACIONES PARA EL LEON 3

Para compilar correctamente los benchmarks utilizados, hicimos uso del compilador de Gaisler Research BCC. Además, para proyectos de más de una fuente disponible en c, fue necesario el uso de la utilidad preparada bajo Eclipse. Este apéndice tiene como función explicar en profundidad los mecanismos necesarios para compilar un archivo c para una arquitectura SPARC v8 como la presente en el Leon 3.

Existen dos maneras de hacerlo, la primera, funcional para un único archivo a compilar, que use las librerías básicas disponibles en la arquitectura SPARC v8, consiste en utilizar la consola de comandos, en concreto, una vez dentro de la carpeta con el archivo a compilar, ejecutamos la siguiente sentencia:

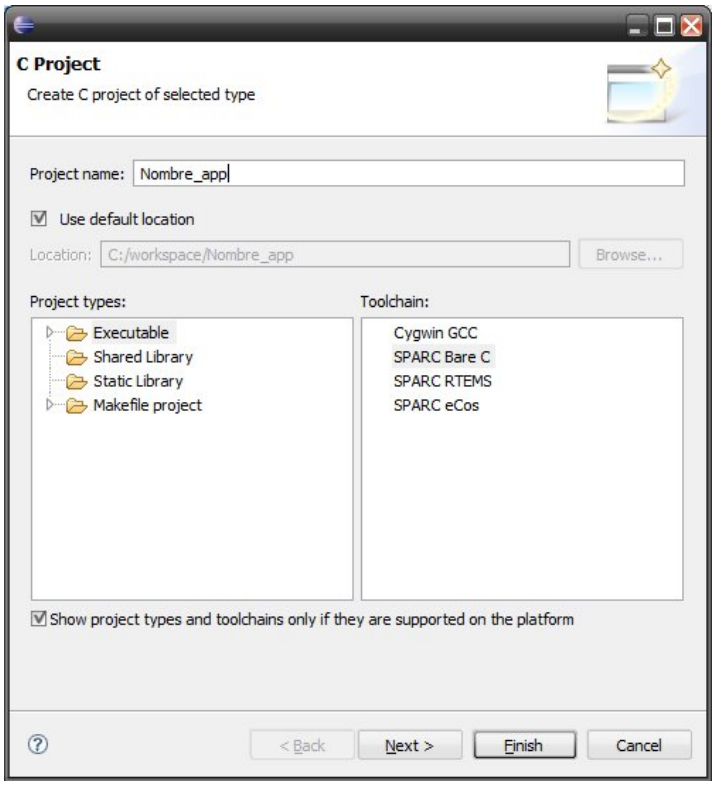
'sparc-elf-gcc -msoft-float -g -OX archivo.c -o archivo.exe'

Donde la X simboliza el nivel de optimización para el compilador, siendo tres tipos los utilizados:

- O0, nivel de optimización: ninguno, el codigo es lo más parecido al escrito. Lo hemos utilizado en las pruebas básicas para comprobar el correcto funcionamiento y puesta en marcha de los predictores.
- O2, optimización orientada al rendimiento, es el adecuado, lo hemos utilizado para comprobar la variación en los resultados de diversos programas. Las versiones finales de los benchmarks se basan en esta versión, de tal manera que son copias exactas de las versiones precompiladas disponibles.
- O3, optimización orientada al espacio, es el utilizado en un principio para compilar los benchmarks, una vez más nos encontramos con los problemas de espacio.

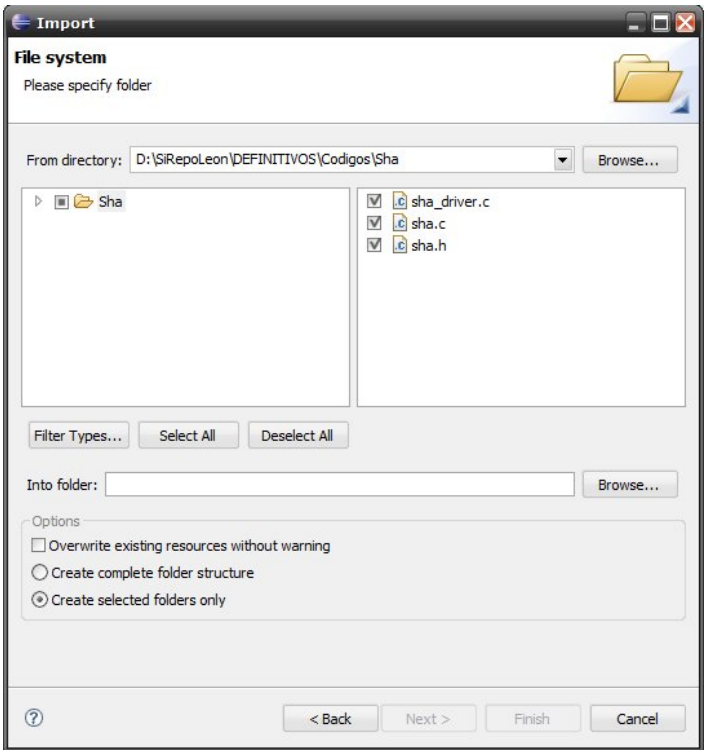
La segunda manera consiste en utilizar la herramienta mencionada bajo eclipse. Presentamos a continuación un breve tutorial de cómo hacerlo, para ello teniendo instalado las GrTools, abrimos el Eclipse, y elegimos File -> New -> C Proyect, y elegimos como opciones de compilación SPARC Bare C y le damos un nombre, este será el mismo que la aplicación final a compilar.

Figura 39. Creación de nuevo proyecto C bajo Eclipse



Tras esto, importamos al proyecto, en caso de necesitarlo, los archivos fuentes necesarios siguiendo el orden siguiente: File -> Import. Una vez aquí, buscamos dentro de la carpeta General, el tipo File System. Tras esto pinchamos en Browse y buscamos la ruta adecuada. En el ejemplo vamos a elegir los archivos fuente necesarios para el benchmark Sha.

Figura 40. Detalle del dialogo para importar archivos fuente bajo Eclipse



Tras esto, ya tenemos los códigos necesarios para trabajar. Una vez hecho los cambios necesarios, en caso de haberlos, para compilar la aplicación abrimos el archivo c que contiene el método main() a compilar, seleccionamos la opción de Release, y pinchamos en el icono adecuado. Tras esto, tendremos la aplicación generada dentro de la carpeta de trabajo, sin extensión y con el nombre del proyecto lista para cargar en el Leon 3.

Figura 41. Boton de generación de aplicacion en Eclipse. Opción Release.